HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Vcred
Date: July 15th, 2022

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 Vcred
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Flashloan
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://vcred.trade/
Timeline	05.05.2022 - 15.07.2022
Changelog	17.05.2022 - Initial Review 26.05.2022 - Second Review 14.06.2022 - Third Review 15.07.2022 - Fourth Review

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

Table of contents

Introduction 4

Scope 4

Severity Definitions 8

Executive Summary 9

Checked Items 10

System Overview 13

Findings 15

Disclaimers 23

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

Introduction

Hacken OÜ (Consultant) was contracted by Vcred (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 is smart contracts in the repository:

Initial review scope
Repository:
https://github.com/VCREDAlpha/flashLoan-a
Commit:
939f76b56d552ef0d417ab084472e87a8403100a
Technical Documentation:
Type: Litepaper
JS tests: Yes
Contracts:
File: ./flashLoan/ArbitrageExecutor.sol
SHA3: d1333d7cf58b703112fa38f5ed3384aaba86b7cf5e63312221cdcae899424a54

File: ./flashLoan/DepositPool.sol
SHA3: 1a70846861512719ad2b8edc66e6ad6703b1a04bc9345d10443da845e120f09c

File: ./flashLoan/FlashBorrower.sol
SHA3: b144b11ab868cbc1fc31cf32ff8ad35848bc797959666f43acb5c153f9a52ec6

File: ./flashLoan/FlashLender.sol
SHA3: 4240dd99fcf3273f67921d3f2c859f4e49732ae45b2fa5fafeea02558cbe96aa

File: ./flashLoan/FlashLoan.sol
SHA3: cd1aa0b90d0b5e334b9f049391906addb6239baaf3bea8d7f1bb415d066786af

File: ./flashLoan/IArbitrageExecutor.sol
SHA3: 63209394a06cb58c48c106303c21def55f96bcdde023a3e4d27bb35d0a0420fd

File: ./flashLoan/IFlashBorrower.sol
SHA3: eb78150f9a91b1d652c98f01a0cb7a85ea811e8978f1895f20211ef95b661578

File: ./reward-distributor/RewardDistributor.sol
SHA3: 9f973b03a62aad16dadf46272fd2aea9bedba03d6023b0f3b08219f4b3cf268c

File: ./reward-distributor/RewardDistributorInterface.sol
SHA3: b9498f61663ac108d8cf77b5b6498c86da1fd01bd6e3941c48be530638f2fc23

File: ./utils/AdminAccess.sol
SHA3: c67225355c46535d8b76ba6c949b3b45988841b645963ce70fc6c20b89cead7b

File: ./vault/AbstractVault.sol
SHA3: 3f83a878bee233fc38193f894787d45df85e73a8ed0f03fcd043bfe4318a9f76

File: ./vault/Vault.sol
SHA3: ae51028c785da86bd46a69630cc0f11f77099dc44fae1c39234b543f478f9476

File: ./vault/VaultInterface.sol
SHA3: e1817882c2694babf85f6cdd68700b070abcbfac83a7af7addabaf18feed4e48

Second review scope

Repository:
https://github.com/VCREDAlpha/flashLoan-a
Commit:
f18ba638ab6984dc6ff55a3b2909c33f2ba361d5
Technical Documentation:
Type: Litepaper
JS tests: Yes
Contracts:
File: ./contracts/AbstractVault.sol
SHA3: 31ce3bcae071fd9e32d2e504ee58bc9040a59c055abf32e8a82e3ea180eab403

File: ./contracts/AdminAccess.sol
SHA3: 7fa549d37133f959be7e2b3b0a5562e0634bbfeb25f52731527212f289d835c6

File: ./contracts/ArbitrageExecutor.sol
SHA3: b5e0f8574cb27f20734e724eb0fb46b610129248e89b1044ca3009351f0f49ea

File: ./contracts/DepositPool.sol
SHA3: 48da1bb20b6c82c058653ea2c4f4e64ed85955fa7926a0edf4fe3ebf4ea5e9fe

File: ./contracts/FlashBorrower.sol
SHA3: 941c1ee9bcbdd4222bf00811a0b8afa50bd83153d2b8d72e3e292522085cecd6

File: ./contracts/FlashLender.sol
SHA3: 27bc472e573f82a2c279e1f6e7cb9339fc8675d1373cab814188337e2d741955

File: ./contracts/FlashLoan.sol
SHA3: 74b108a46ef8fdf255a6efd15788c7882a6aa1d4b996aaffc72838388b7c6003

File: ./contracts/IArbitrageExecutor.sol
SHA3: 66c31589ada02431e3d2fafc439da5898e991488711198ee0ad5a676231f4462

File: ./contracts/IFlashBorrower.sol
SHA3: 6feea8b0157ea27dd567a03ab4ca0ac8449da3f8b7224aec3106d6affa5f4d46

File: ./contracts/RewardDistributor.sol
SHA3: 37c85104c6c44af8180b0413bd4cb008961f150c85335026a1dab6c04c2a39d0

File: ./contracts/RewardDistributorInterface.sol
SHA3: 460162bf81a276836d83bf393434a91256d60843ea9f7d798d3a3320b06fbd3b

File: ./contracts/Vault.sol
SHA3: 98ae93e7741f9f7b8f7e8dbe1cc26481cf81e71a22194794828e4fb90bf2a5ea

File: ./contracts/VaultInterface.sol
SHA3: f898c2236a7bf508c8642ff930bdbf7bf751fec34ba9f3ae75dd8003284be09b

Third review scope

Repository:
https://github.com/VCREDAlpha/flashLoan-a
Commit:
f18ba638ab6984dc6ff55a3b2909c33f2ba361d5
Technical Documentation:
Type: Litepaper
JS tests: Yes
Contracts:
File: ./contracts/AbstractVault.sol
SHA3: 31ce3bcae071fd9e32d2e504ee58bc9040a59c055abf32e8a82e3ea180eab403

File: ./contracts/AdminAccess.sol
SHA3: 7fa549d37133f959be7e2b3b0a5562e0634bbfeb25f52731527212f289d835c6

File: ./contracts/ArbitrageExecutor.sol
SHA3: b5e0f8574cb27f20734e724eb0fb46b610129248e89b1044ca3009351f0f49ea

File: ./contracts/DepositPool.sol
SHA3: de91cc5818e2c03cfe517975651c2201b5b944cf634cdd6e9683041bc0f14d18

File: ./contracts/FlashBorrower.sol
SHA3: 941c1ee9bcbdd4222bf00811a0b8afa50bd83153d2b8d72e3e292522085cecd6

File: ./contracts/FlashLender.sol
SHA3: cdf68dfc6a43b514b54cb8632384ad92f6c9a382d3ec2aebd94b86faa3b53e47

File: ./contracts/FlashLoan.sol
SHA3: 74b108a46ef8fdf255a6efd15788c7882a6aa1d4b996aaffc72838388b7c6003

File: ./contracts/IArbitrageExecutor.sol
SHA3: 66c31589ada02431e3d2fafc439da5898e991488711198ee0ad5a676231f4462

File: ./contracts/IFlashBorrower.sol
SHA3: 6feea8b0157ea27dd567a03ab4ca0ac8449da3f8b7224aec3106d6affa5f4d46

File: ./contracts/RewardDistributor.sol
SHA3: 0d3f13c65b857046bc0b9e52c440b01a364cf5304709407af7ce98a798341f1f

File: ./contracts/RewardDistributorInterface.sol
SHA3: dd50192655504e9fe17a10aba6fa0a6e5b76264871907a6c6408e550adc1ce44

File: ./contracts/Vault.sol
SHA3: 98ae93e7741f9f7b8f7e8dbe1cc26481cf81e71a22194794828e4fb90bf2a5ea

File: ./contracts/VaultInterface.sol
SHA3: f898c2236a7bf508c8642ff930bdbf7bf751fec34ba9f3ae75dd8003284be09b

Fourth review scope

Repository:
https://github.com/VCREDAlpha/flashLoan-a

Commit:
e2958492007f6ae9c5cac7ac4058b25621e1c2be

Technical Documentation:
Type: Litepaper

Type: Technical description

JS tests: Yes

Contracts:
File: ./contracts/ArbitrageExecutor.sol
SHA3: a0a6e8fd40a9d2f6d1e12311bb1e34854a77f76cd7931f0c2aa2e23e76da1cc8

File: ./contracts/DepositPool.sol
SHA3: 50059e606fe88a67779bcd9362085fc0b4baa2bcfb8e58a928ee940c0eb966db

File: ./contracts/FlashBorrower.sol
SHA3: a470b4b727a6be3a60b0c53ec720d3be7f2cea85a51dca96e535901ed259c0d3

File: ./contracts/FlashLender.sol
SHA3: 329267c52adca628a53f57b47961c7a4cac83a496633530a891a28df5d0a9f9e

File: ./contracts/IArbitrageExecutor.sol
SHA3: d7f81a66e1e1a8efb68c82cd73e708cbc5917b4d52658ece623e0dfac20cc938

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

```
File: ./contracts/IFlashBorrower.sol
SHA3: 16d9c444b770800ed866c20db795a78d06ec6b7248871e7394c7acc5ea92b353

File: ./contracts/reward-distributor/RewardDistributor.sol
SHA3: 5c2212a118c2ad4c0d44e9ddb8ffc35414a1e8fb7070208b334e10496f18de8b

File: ./contracts/reward-distributor/RewardDistributorInterface.sol
SHA3: 1b8b1456c4e049cdbae44bed5b37d30fc6bb80180f83b4ff66f502b94170d3fa

File: ./contracts/reward-distributor/Vault.sol
SHA3: 7fd2d415b6ffa6433f54a213283803b9ae16665dccaf7c0aafb21bbdb80b4487

File: ./contracts/reward-distributor/VaultInterface.sol
SHA3: 9c23de7540b92fbac48c2834bb47e5b851c0bf29847fb4492faa17bd1a39fa57
```

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 assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 methodology.

Documentation quality

The Customer provided functional requirements and a technical description. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. Most of the code follows official language style guides. Unit tests were provided.

Architecture quality

The architecture quality score is **5** out of **10**. The development environment is provided. The code contains circular dependencies.

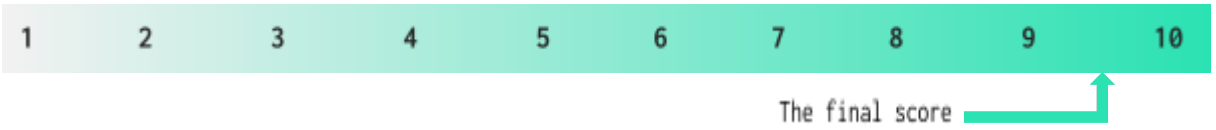
Security score

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: **9.5**.



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

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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 destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
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 it is required.	Passed
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.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	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
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.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, 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, that may be changed in the future.	Passed

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

System Overview

VCRED is a flashloan system with the following contracts:

- *FlashBorrower* – is a contract for borrowing funds. It allows to borrow native and not native tokens. When calling borrow, calls the *FlashLender* contract, executes arbitrage with borrowed funds using *ArbitrageExecutor* contract, and returns funds to the *FlashLender* contract. (Transfers for native tokens, approves for not native tokens) Has a whitelist functionality.
- *FlashLender* – is an abstract contract for lending funds. Allows to flashloan native and not native whitelisted tokens. Each *FlashLender* has a *RewardDistributor* and *Vault*. After the debts are repaid, extra funds are distributed to the *RewardDistributor*.
- *DepositPool* – is a flash lender (inherits from the *FlashLender* contract). Allows to deposit native and not native whitelisted tokens and withdraw them with the rewards or separately.
- *ArbitrageExecutor* – executes arbitrage using Uniswap router. Swaps token A (native if arbitrage native) for token B, then swaps token B for token A. Returns the result to the FlashBorrower.
- *RewardDistributor* – is a contract that allows to distribute rewards. Uses *Vault* to transfer tokens, uses chainlink to get tokens prices. The rewards depend on the deposited amount: after the debts are repaid, extra funds are distributed to the contract and divided by the total deposited amount. The rewards are paid in the stablecoin defined by the *Vault* owner.
- *Vault* – is a contract used for saving and transferring rewards tokens. The tokens and native coins are stored in the contract, the owner can withdraw them and deposit the stablecoin that is used for rewards payment to the contract.
- *IArbitrageExecutor* – is an *ArbitrageExecutor* interface.
- *IFlashBorrower* – is an *FlashBorrower* interface.
- *RewardDistributorInterface* – is a *RewardDistributor* interface.
- *VaultInterface* – is a *Vault* interface.

Privileged roles

- The owner of the *FlashBorrower* contract can add and remove users from the whitelist, set and update the arbitrage executor address.
- The owner of the *FlashLender* contract can set and update *FlashBorrower* address, add, remove the whitelist tokens and update their parameters (addresses, limits, and fees percentages for borrowing).
- The owner of the *RewardDistributor* contract can deposit, withdraw stakes and rewards, and distribute, add and remove admins. It is implied that the owner is a *FlashLender* contract.
- The owner of the *Vault* contract can withdraw and deposit tokens.

- The admin of the *Vault* contract can add and remove the *RewardDistributor.*
- The RewardDistributor of the *Vault* contract can transfer tokens from the contract.

Risks

- The project contains contracts not included in the audit scope that cannot be verified.
- Rewards payment can not be guaranteed as rewards are transferred by the *Vault* owner.
- After the stake withdrawal, it is not possible to get the reward for it, rewards withdrawal should be made first.

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

Findings

Critical

1. Incorrect imports

"./DSMath.sol" for DepositPool.sol, FlashBorrower.sol, FlashLender.sol is incorrect, there is no “DSMath” file in the project.
"pull-based-reward-distribution/contracts/RewardDistributorInterface.sol",
"pull-based-reward-distribution/contracts/RewardDistributor.sol" for FlashLender.sol, 'vault/contracts/VaultInterface.sol' for RewardDistributor.sol, 'utils/contracts/access/AdminAccess.sol' for Vault.sol have incorrect paths. Missed SafeMath import in FlashLender.sol.

Therefore, the contracts cannot be compiled.

Contracts: DepositPool.sol, FlashBorrower.sol, FlashLender.sol, RewardDistributor.sol, Vault.sol

Function: -

Recommendation: replace the incorrect paths in imports with the correct ones and remove the imports of missing files.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

2. Unsafe int to uint converting

When calculating the reward in the “computeReward” function, the value of “rewardTally[account]” is subtracted from the value of “actualReward”. If the “rewardTally[account]” is bigger than “actualReward”, negative “reward” value will be converted to uint256 in the “withdrawReward” function. The “reward” value in the “distribute” function is not checked if it is bigger than 0.

This may lead to unexpected behavior: users may withdraw incorrect reward amounts.

Contract: RewardDistributor.sol

Functions: withdrawReward, computeReward, distribute

Recommendation: ensure that “reward” values in “withdrawReward” and “distribute” functions are not negative.

Status: Fixed (Revised commit: e2958492007f6ae9c5cac7ac4058b25621e1c2be)

High

1. Highly permissive owner access

The functionality allows the owner to withdraw any amount of tokens from the contracts.

The users` funds may be withdrawn.

Contracts: FlashLender.sol, DepositPool.sol

Functions: directWithdrawNative, directWithdraw

Recommendation: deprive the owner of the ability to withdraw tokens.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

2. **Missing of time dependence in the rewards paying**

Stacking time is not considered to determine the possibility of getting the reward.

This may lead to the flashloan attack: depositing flashloaned assets and getting rewards.

Contract: RewardDistributor.sol

Function: withdrawReward

Recommendation: do not allow depositing funds and getting rewards in one block. (Checking if block.number is the same).

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

3. **Pausing functionality**

The functionality allows the owner to pause the contract.

Users will not be able to withdraw their funds and rewards.

Such behavior is not described in the documentation, or documentation was not provided.

Contract: RewardDistributor.sol

Functions: withdraw, withdrawStake, withdrawReward

Recommendation: remove the pausing functionality or describe this behavior in the publicly available documentation.

Status: Fixed (Revised commit: f18ba638ab6984dc6ff55a3b2909c33f2ba361d5)

4. **Highly permissive owner access**

The owner can withdraw any tokens from the contract.

The users` funds (rewards) may be withdrawn.

Contract: Vault.sol

Functions: *In the first-third review:* withdrawToken, withdraw. *In the fourth review:* retrieveCollected, retrieveCollected

Recommendation: do not withdraw tokens that belong to users.

Status: Mitigated. The functionality is implied by design.

5. **TODO notices**

   There are TODO notices in the contract.

   This can indicate that the code is not finalized.

   Contract: FlashLender.sol

   Functions: directWithdraw, directDeposit, directWithdrawNative, directDepositNative

   Recommendation: resolve the TODO issues.

   Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

6. **Highly permissive owner access**

   The deposited tokens can be withdrawn if the token is whitelisted.

   This may block the withdrawal of users` funds if the owner removes the token from the whitelist.

   Contract: DepositPool.sol

   Function: withdraw

   Recommendation: ensure that users can withdraw funds.

   Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

7. **Insufficient funds**

   Rewards are paid to users during the unstake process.

   If the rewards can not be paid, all transactions will fail, and users will not be able to withdraw funds.

   Contract: RewardDistributor.sol

   Function: withdraw

   Recommendation: allow users to withdraw funds at any time.

   Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

8. **Loss of rewards**

   withdraw function gives back the staked amount whether the reward is distributed.

   If there is not enough reward token in the contract, users who want to withdraw rewards and staked tokens can accidentally unstake the tokens without getting the reward.

   Contract: RewardDistributor.sol

   Function: withdraw

Recommendation: put an emergency function for users to withdraw staked tokens if there is no reward token or fix the logic differently.

Status: Fixed (Revised commit: f18ba638ab6984dc6ff55a3b2909c33f2ba361d5)

Medium

1. Check-Effect-Interaction pattern violation

The state variables are changed after the external calls are executed.

Executing external calls after changing the state protects the contract from re-entrance and race conditions if calling ambiguous contracts.

Contract: DepositPool.sol

Functions: depositeNative, withdrawNative, deposit, withdraw

Recommendation: ensure all state changes are performed before the external calls are executed or use a reentrancy lock.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

2. Redundant arrays usage as the function parameters

The “_tokens” and “_contracts” arrays are used as the parameters, though only the first two elements of arrays are used.

Contracts: FlashBorrower.sol. FlashLender.sol, ArbitrageExecutor.sol

Functions: FlashBorrower.executeOnFlashLoanNative, FlashBorrower.executeOnFlashLoan, ArbitrageExecutor.exec, ArbitrageExecutor.execNative, FlashLender.FlashLoan, FlashLender.FlashLoanNative

Recommendation: replace the arrays with the separate variables. For example, tokenA, tokenB, contractA, contractB.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

Low

1. Using state variables default visibility

The visibilities of FlashLender.nativeDistributor, FlashLender.nativeTokenAddress, FlashLender.flashBorrower, ArbitrageExecutor.wCoin, AdminAccess.administrators, RewardDistributor.multiplier variables are not labeled.

It is recommended to label visibility to catch incorrect assumptions about who can access the variable.

Contracts: FlashLender.sol, ArbitrageExecutor.sol, AdminAccess.sol, RewardDistributor.sol

Function: -

Recommendation: explicitly mark visibility for state variables.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

2. **Unlocked pragma**

Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs.

Contracts should be deployed with the same compiler version they have been tested thoroughly.

Contracts: FlashLender.sol, DepositPool.sol, FlashBorrower.sol, IFlashBorrower.sol, ArbitrageExecutor.sol, IArbitrageExecutor.sol, FlashLoan.sol, AdminAccess.sol, RewardDistributor.sol, RewardDistributorInterface.sol, Vault.sol, AbstractVault.sol, VaultInterface.sol

Function: -

Recommendation: lock pragma to a specific compiler version.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

3. **Using SafeMath**

SafeMath is generally not needed starting with Solidity 0.8.

Contracts: DepositPool.sol, FlashBorrower.sol, FlashLender.sol, Vault.sol

Function: -

Recommendation: remove SafeMath.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

4. **Redundant pragma experimental ABIEncoderV2 defining**

ABI coder v2 is activated by default, starting with Solidity 0.8.

Contract: FlashBorrower.sol

Function: -

Recommendation: remove ABI coder v2 defining.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

5. **Redundant imports**

Imports of "hardhat/console.sol" are redundant.

Contracts: Vault.sol, FlashBorrower.sol, FlashLender.sol, RewardDistributor.sol

Function: -

Recommendation: remove the redundant imports.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

6. **Functions that can be declared as external**

There are public functions in the contracts that are not used internally.

“External” visibility uses less Gas.

Contracts: DepositPool.sol, FlashBorrower.sol, FlashLender.sol

Functions: DepositPool.withdrawNative, DepositPool.deposit, DepositPool.withdraw, FlashBorrower.addWhiteListedAccount, FlashBorrower.removeWhiteListedAccount, FlashBorrower.arbitrageExecutor, FlashBorrower.flashLender, FlashBorrower.borrow, FlashBorrower.borrowNative, FlashLender.balanceOf, FlashLender.balanceOfNative, FlashLender.rewardOf, FlashLender.rewardOfNative

Recommendation: replace the visibilities to “external”.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

7. **Redundant variable**

The contract contains “ONE” variable that is never used.
Contract: FlashBorrower.sol

Function: -

Recommendation: remove the redundant variable.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

8. **Not enough information in the event**

The information in the “Withdraw” event is not full because users can withdraw different amounts of tokens.

Contract: DepositPool.sol

Function: -

Recommendation: add the information about the amount of withdrawn tokens to the “Withdraw” event.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

9. **Redundant functions parameters**

The “debt” parameters are not used.

Contracts: FlashBorrower.sol, IFlashBorrower.sol

Functions: executeOnFlashLoan, executeOnFlashLoanNative

Recommendation: remove the unused parameters.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

10. No events on state variables changes

There are no events emitted on important state variables changes.

Contracts: FlashLender.sol, Vault.sol

Functions: FlashLender.setBorrower, FlashLender.addWhiteListToken, FlashLender.removeWhiteListToken, FlashLender.addNativeRewardDistributor, FlashLender.addRewardDistributor, FlashLender.updateCoin, FlashLender.updateToken, FlashLender. directWithdraw, FlashLender. directDeposit, FlashLender.directWithdrawNative, FlashLender.directDepositNative, Vault.deposit, Vault.withdraw, Vault.withdrawToken

Recommendation: emit events on important state variables changings.

Status: Fixed (Revised commit: 93b75351d6838bb56f64e90c52f683b13e833733)

11. Choosing wrong variable type

In the Token struct, _active variable is declared as uint256. It can have two different values; active or not.

It can cause unnecessary Gas consumption and a decrease in code readability.

Contract: FlashLender.sol

Function: -

Recommendation: declare it as a boolean variable.

Status: Fixed (Revised commit: f18ba638ab6984dc6ff55a3b2909c33f2ba361d5)

12. Redundant modifier usage

Using “tokenAdded” modifier is redundant, because there is “onlyWhitelistedToken” modifier check.

Contract: DepositPool.sol

Function: deposit

Recommendation: remove the redundant modifier usage.

Status: Fixed (Revised commit: f18ba638ab6984dc6ff55a3b2909c33f2ba361d5)

13. Duplicated code

The “withdrawNative”, “withdrawNativeStake” and “withdraw”, “withdrawStake” functions use the same code.

Contract: DepositPool.sol

Functions: withdrawNative, withdrawNativeStake, withdraw, withdrawStake

Recommendation: put the duplicated code in a separate function.

Status: Fixed (Revised commit: e2958492007f6ae9c5cac7ac4058b25621e1c2be)

14. Redundant import

The "@openzeppelin/contracts/security/Pausable.sol" import is not used.

Contract: RewardDistributor.sol

Function: -

Recommendation: remove the redundant import.

Status: Fixed (Revised commit: e2958492007f6ae9c5cac7ac4058b25621e1c2be)

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 by the best industry practices at the date 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.