WEICHAIN Security Assessment Certificate

2nd of June 2023 — WeiChain Verified

Return Finance

This smart contract audit was prepared by WeiChain.

Post-Audit Conclusion

The **Return Finance** team remediated all exhibits outlined in the report and the code quality is excellent. The codebase adheres to the best practices and standards of smart contract development. During the audit process, our team thoroughly reviewed the smart contract code, conducted various tests, and analyzed the contract's functionality and security aspects. We initially identified several areas that required attention, including potential vulnerabilities and code inefficiencies. However, we are pleased to note that all these issues have been successfully resolved and acknowledged.

The improvements made to the smart contract code are commendable. The codebase demonstrates a robust structure, with clear and concise logic that is easily understandable. The contract's functionality has been thoroughly tested, ensuring that it performs as intended and meets the specified requirements. Furthermore, the contract implements essential security measures, mitigating potential risks and vulnerabilities.

The codebase can be considered of a high quality. It exhibits a high level of clarity, with well-commented sections and consistent naming conventions, making it easy to maintain and understand. The use of standardized libraries and frameworks has enhanced code readability and reduced the likelihood of introducing bugs or vulnerabilities. Additionally, the contract's implementation aligns with industry best practices, promoting efficiency and scalability.

Overall, we are confident in stating that the smart contract of **Return Finance**, has successfully passed the post-audit evaluation. The codebase is of high quality, following best practices, and all identified issues have been resolved. We commend your commitment to ensuring a secure and well-developed smart contract.

Executive Summary

Type	Smart Contracts
Auditors	Krasimir Raykov
Timeline	2023-05-29 through 2023-06-02
EVM	Shanghai
Languages	Solidity
Methods	Architecture Review, Computer-Aided Verification, Manual Review
Specifications	README.md

Total issues	10
Critical	0
Medium	4
Low	1
Informational	5

Critical	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for clients and users.
Medium	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The scope of the audit is restricted to the set of files outlined in the Audit Breakdown section. While reviewing the given files at the commit `c1cd40aa0a6958d15e19b0fbda8566838d02a358`, we identified **0 issue of critical severity, 4 issues of medium severity, 1 issue of low severity and 5 issues of informational severity.**
In addition, we made several suggestions with regards to code documentation, adherence to best practices, and adherence to the specification. We recommend resolving the issues and improving code documentation before shipping to production.

The issues were resolved in commit `72e2bfb0fd44560722f6e6187da75980647fab88`

ID	Description	Severity	Status
WCH – 1	Use SafeERC20 library	Medium	Resolved
WCH – 2	Overpowered role	Low	Acknowledged
WCH – 3	Consider using `constant` instead of `immutable`	Informational	Resolved
WCH – 4	Changing the state within a function modifier	Informational	Resolved
WCH – 5	Potentially misleading behavior	Informational	Acknowledged
WCH – 6	Omit block.timestamp from events	Informational	Resolved
WCH – 7	Withdraw more funds than intended	Medium	Resolved
WCH – 8	Possible unexpected behavior	Medium	Resolved
WCH – 9	Ignored return values	Informational	Acknowledged
WCH – 10	Security concern: Inflation attack	Medium	Resolved

Code Coverage and inline documentation

The contracts that are in-scope are well covered with tests. There is 85% function coverage and 63% branch coverage. We strongly advise the implementation of comprehensive unit tests, with the goal of achieving a minimum code coverage of 90%

The source-units contain inline documentation which makes it easier to reason about methods and how they are supposed to be used.

Static Analysis

The execution of our static analysis toolkit identified **103 potential issues** within the codebase of which **95 were ruled out to be false positives** or negligible findings.

The remaining **issues** were validated and grouped and formalized into the **1 exhibits** - WCH – 9

Audit Breakdown

WeiChain's objective was to evaluate the following files for security-related issues, code quality, and adherence to specification and best practices:

* ReturnFinanceUSDCVault.sol
* IAaveV3Pool.sol
* ICompoundUSDCV3.sol
* IConicLpTokenStaker.sol
* IConicOmniPool.sol
* IConicRewardManager.sol
* ICurvePoolV2.sol
* ISwapRouter.sol
* IYearnVault.sol

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The WeiChain auditing process follows a routine series of steps:

1. Code review that includes the following
    1.1. Review of the specifications, sources, and instructions provided to WeiChain to make sure we understand the size, scope, and functionality of the smart contract.
    1.2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    1.3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to WeiChain describe.
2. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
3. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

WCH-1 Use SafeERC20 library

Severity: Medium

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: Some ERC20 tokens revert on failure, but doesn't return `true` on success. Quite a few tokens are affected including big names like USDT and BNB.

- ReturnFinanceUSDCVault.sol, L345: `sweepFunds()`

Recommendation: Use SafeERC20 library.

WCH-2 Overpowered role

Severity: **Low**

Status: Acknowledged

File(s) affected: ReturnFinanceUSDCVault.sol

Description: There are functions callable only from one address. Therefore, the system depends heavily on this address. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the private key of this address becomes compromised.

Recommendation: Use multisignature wallet for the owner and implement timelock with at least 3 days timelock period. (https://blog.openzeppelin.com/protect-your-users-with-smart-contract-timelocks)

WCH-3 Consider using `constant` instead of `immutable`

Severity: **Informational**

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: Immutable state can be initialized when the contract is deployed (in the constructor). There are multiple state variables that are declared as immutable, but are not initialized in the constructor.

Recommendation: Use `constant` instead of `immutable` for state variables that aren't initialized in the constructor.

WCH-4 Changing the state within a function modifier

Severity: **Informational**

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: It is generally not recommended to write logic that changes the state within a function modifier. Function modifiers are intended to modify the behavior of functions without altering their state-changing functionality. Modifiers are commonly used for access control, input validation, or logging.

By convention, function modifiers should not modify the state of the contract, as it can lead to unexpected behavior and make the code harder to reason about. Modifiers should focus on enforcing conditions or adding additional checks before executing the function.

Recommendation: Call `harvestConicRewardsAndSwapForUnderlying` explicitly and not in modifier

WCH-5 Potentially misleading behavior

Severity: Informational

Status: Acknowledged

File(s) affected: ReturnFinanceUSDCVault.sol

Description: The method `harvestConicRewardsAndSwapForUnderlying` is public and can be called by a user. The user might expect to receive the underlying token after calling this function.

Recommendation: Change the visibility modifier for `harvestConicRewardsAndSwapForUnderlying`

WCH-6 Omit block.timestamp from events

Severity: Informational

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: When fetching events off-chain, information about the block number is available, allowing you to retrieve the block.timestamp. However, including the block.timestamp in the event data would redundantly increase the overall transaction fees of the contract.

Recommendation: Remove `block.timestamp` from all events.

WCH-7 Withdraw more funds than intended

Severity: Medium

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: If the balance of the underlying token is insufficient, the withdraw method will withdraw the remaining amount from the pools. Subsequently, a check is performed to ensure that the total token balance is greater than `minAmountOut`. However, due to slippage, there is a possibility that the withdrawal amount exceeds the requested amount. This could lead to users withdrawing more funds than intended.

Recommendation: Additional check against `previewAmount` or pass `previewAmount` to the internal `_witdhraw` method

WCH-8 Possible unexpected behavior

Severity: Medium

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: Currently `rescueFunds` tries to withdraw funds from four external contracts. If any of the external calls fails, none of the withdrawals would succeed. Same applies for the methods `_withdrawFromPools & _depositToPools.` This could lead to funds being stuck.

Recommendation: Use try/catch for each withdrawal.

WCH-9 Ignored return values

Severity: Informational

Status: Acknowledged

File(s) affected: ReturnFinanceUSDCVault.sol

Description: The return value of an external call is not stored in a local or state variable. For example, `_withdrawFromPools` does not check the return value when making an external call `unstakeAndWithdraw`. If the internal logic of the external contract does not check if the minimum requirements for withdraw are met, then this could lead to loss of funds.

Recommendation: It is important to consistently consider the possibility of abnormal behavior in the external logic being called, and to ensure that your internal business logic operates correctly. To achieve this, it is crucial to verify the return value of all external calls.

WCH-10 Security concern: Inflation attack

Severity: **Medium**

Status: Resolved

File(s) affected: ReturnFinanceUSDCVault.sol

Description: When depositing tokens, the number of shares a user gets is rounded down. This rounding takes away value from the user in favor or the vault (i.e. in favor of all the current share holders). This rounding is often negligible because of the amount at stake. If you deposit 1e9 shares worth of tokens, the rounding will have you lose at most 0.0000001% of your deposit. However if you deposit 10 shares worth of tokens, you could lose 10% of your deposit. Even worse, if you deposit <1 share worth of tokens, then you get 0 shares, and you basically made a donation.

For a given amount of assets, the more shares you receive the safer you are. If you want to limit your losses to at most 1%, you need to receive at least 100 shares. The idea of an inflation attack is that an attacker can donate assets to the vault to move the rate curve to the right, and make the vault unsafe.

Recommendation: Defending with a virtual offset. Since OpenZeppelin v4.9, the ERC4626 implementation uses virtual assets and shares to mitigate that risk. Currently, the project uses version 4.3.2, to mitigate that risk you should consider updating to 4.9

Disclaimer