7th of February 2024 — WeiChain Verified

Return Finance v2

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 **fixed** or **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	2024-30-Jan through 2024-07-Feb
EVM	Paris
Languages	Solidity
Methods	Architecture Review, Computer-Aided Verification, Manual Review
Specifications	N/A

Total issues	9
High	3
Medium	3
Low	3

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **Critical** - client must fix the issue.
- **High** - client must fix the issue.
- **Medium** - client should fix the issue.
- **Low** - client could fix the issue

Summary of Findings

The scope of the audit is restricted to the set of files outlined in the Audit Breakdown section.

ID	Description	Severity	Status
WCH – 1	First ERC4626 deposit can break share calculation	Medium	**Acknowledged**
WCH – 2	The protocol uses `_msgSender()` extensively, but not everywhere	Medium	**Fixed**
WCH – 3	ERC4626 does not work with fee-on-transfer tokens	Low	**Acknowledged**
WCH – 4	Chainlink's `latestRoundData` might return stale or incorrect results	Medium	**Fixed**
WCH – 5	Approving MAX_UINT amount of ERC20 tokens	Low	**Fixed**
WCH-6	`callExternalContract`, ` sweepFunds` and `rescueFunds` can be used to steal user's funds	High	**Acknowledged**
WCH-7	`calc_withdraw_one_coin` is vulnerable to manipulation	High	**Fixed**
WCH-8	High Solidity compiler optimizations can be problematic	Low	**Fixed**
WCH-9	Uninitialized state variable	High	**Fixed**

Code Coverage and inline documentation

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

Static Analysis

The execution of our static analysis toolkit identified **117 potential issues** within the codebase of which **116 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 at commit 0b517c81c99733027a7976dbb371070dd9b21a2b for security-related issues, code quality, and adherence to specification and best practices:

All fixed issues can be found at commit f3a3e58ee000649017b44f0d97885beba0a3c0fb

ReturnFinanceAaveV3USDCVault.sol
ReturnFinanceCompoundV3USDCVault.sol
ReturnFinanceConvexUSDCVault.sol
ReturnFinanceCurveEUROCVault.sol
ReturnFinanceSparkUSDCVault.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 First ERC4626 deposit can break share calculation

Status: Acknowledged: Only whitelisted addresses can deposit and the team will make sure that the vaults aren't empty.

Impact: High

Likelihood: Low

Description: The first depositor of an ERC4626 vault can maliciously manipulate the share price by depositing the lowest possible amount (1 wei) of liquidity and then artificially inflating ERC4626.totalAssets.
This can inflate the base share price as high as 1:1e18 early on, which force all subsequence deposit to use this share price as a base and worst case, due to rounding down, if this malicious initial deposit front-run someone else depositing, this depositor will receive 0 shares and lost his deposited assets.

Recommendation: This is a well-known issue, Uniswap and other protocols had similar issues when supply == 0.
For the first deposit, mint a fixed amount of shares, e.g. 10**decimals()

```
if (supply == 0) {
    return 10**decimals;
} else {
    return assets.mulDivDown(supply, totalAssets());
}
```

WCH-2 The protocol uses `_msgSender()` extensively, but not everywhere

Status: Fixed

Impact: Low

Likelihood: High

Description: The code is using OpenZeppelin's Context contract which is intended to allow meta-transactions. It works by using doing a call to _msgSender() instead of querying msg.sender directly, because the method allows those special transactions. The problem is that the **_swapExactInputMultihop** method in ReturnFinanceCurveEUROCVault use msg.sender directly instead of _msgSender(), which breaks this intent and will not allow meta-transactions.

Recommendation: Change the code in the **swapExactInputMultihop** method to use _msgSender() instead of msg.sender.

WCH-3 ERC4626 does not work with fee-on-transfer tokens

Status: Acknowledged: No fee-on-transfer tokens are currently supported.

Impact: Low

Likelihood: Low as currently all supported tokens doesn't have fee on transfer

Description: The ERC4626 deposit/mint functions do not work well with fee-on-transfer tokens as the assets variable is the pre-fee amount, including the fee, whereas the totalAssets do not include the fee anymore.

Recommendation: Maybe previewDeposit should be overwritten by vaults supporting fee-on-transfer tokens to predict the post-fee amount. If not, then the code must be updated if one of the tokens (e.g. USDC) have an upgrade to include fee on transfer.

WCH-4 Chainlink's `latestRoundData` might return stale or incorrect results

Status: Fixed

Impact: High

Likelihood: Low

Description: The **crvPriceEUR**, **crvPriceUSD** and **cvxPriceUSD** methods call out to a Chainlink oracle receiving the latestRoundData(). If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started).

Recommendation: Add the following checks:

```
...
( roundId, rawPrice, , updateTime, answeredInRound ) = AggregatorV3Interface(XXXXX).latestRoundData();
require(rawPrice > 0, "Chainlink price <= 0");
require(updateTime != 0, "Incomplete round");
require(answeredInRound >= roundId, "Stale price");
...
```

WCH-5 Approving MAX_UINT amount of ERC20 tokens

Status: Fixed

Impact: Medium

Likelihood: Low

Description: The contracts approves the max amount of the supported tokens instead of just approving the specific amount when needed to be sent. This behavior could lead to security issues and potential losses in the case of a security breach in any of the external contracts.

Recommendation: It is recommended to approve only the minimum amount necessary, or set approval to zero after the operations for a contract to perform its intended functions. This minimizes risks, reduces potential losses in case of a security breach, and follows the principle of least privilege.

WCH-6 `callExternalContract`, ` sweepFunds` and `rescueFunds` can be used to steal user's funds

Status: Acknowledged: This level of centralization is acceptable.

Impact: High

Likelihood: Low

Description: `callExternalContract`, ` sweepFunds` and `rescueFunds` can be used by the owner of the contracts to steal user funds.

Recommendation: Add a time-lock with a reasonable lock period to mitigate abuses in case of stolen private keys. Reference: Contract module which acts as a timelocked controller.

WCH-7 `calc_withdraw_one_coin` is vulnerable to manipulation

Status: Fixed

Impact: High

Likelihood: Medium

Description: The **totalAssets** method in **ReturnFinanceConvexUSDCVault.sol** uses **calc_withdraw_one_coin** function from Curve. This method is a Curve AMM function based on the current state of the pool and changes as trades are made through the pool. This spot price can be manipulated using a flash loan or large trade.

Recommendation: Short term, do not use the Curve AMM spot price to get the total assets. Long term, use an oracle or get_virtual_price to reduce the likelihood of manipulation. Reference: Economic Attack on Harvest Finance— Deep Dive

WCH-8 High Solidity compiler optimizations can be problematic

Status: Fixed

Impact: High

Likelihood: Medium

Description: The optional compiler optimizations in Solidity is currently set to 9999 which is considered very high.

There have been several optimization bugs with security implications. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised.

High-severity security issues due to optimization bugs have occurred in the past. A high-severity bug in the emscripten-generated solc-js compiler used by Truffle and Remix persisted until late 2018. The fix for this bug was not reported in the Solidity CHANGELOG. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6. More recently, another bug due to the incorrect caching of keccak256 was reported.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe.

It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations.

Recommendation: Short term, measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug. Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity.

WCH-9 Uninitialized state variable

Status: Fixed

Impact: Medium

Likelihood: High

Description: The state variable `ageur` is never initialized, but it's used in **ReturnFinanceCurveEUROCVault::afterDepositOrWithdraw()**

Recommendation: Initialize the state variable.

Disclaimer

WeiChain audit is not a security warranty, investment advice, or an endorsement of ReturnFinance or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

WeiChain Ltd.