CERTIK

Code Security Assessment

Rift Finance

Feb 4th, 2022

Table of Contents

Summary

This report has been prepared for Rift Finance to discover issues and vulnerabilities in the source code of the Rift Finance project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Rift Finance
Platform	ethereum
Language	Solidity
Codebase	• https://github.com/Rift-Finance/rift-protocol
Commit	• 753cdb4088545d312e1699e7bdc42402c9678113 • 1ab54221a7266331a6fa17cb1d9da5d028a14814 • 53e5a59afd99fabeed0fa1f21bba573d650049eb

Audit Summary

Delivery Date	Feb 04, 2022
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
Critical	0	0	0	0	0	0	0
Major	3	0	0	3	0	0	0
Medium	0	0	0	0	0	0	0
Minor	0	0	0	0	0	0	0
Informational	9	0	0	1	0	0	8
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
CCK	core/Core.sol	e42a1f9916b061bcb0c7dda0c21bae5f6154f8c2ef494571b418278504204c72
CPC	core/CorePermissions.sol	e46d3f2418f76c1f59ed1abefae9a9044a0d991f428699d1c838efb0026fd89a
CSC	core/CoreStorage.sol	df457dc0c4b6b11e537ac4ac5e581a4af2cd3b42fea2bb399d6994713d5aa1f1
ICC	core/ICore.sol	43212645f2c0c2448bef853707a70f11ab089f72c6084d089efc219c4829a400
ICP	core/ICorePermissions.sol	731a1471fa39c9bc393a29476bf407c15098362390456aa51cefc1995911a576
IMC	external/sushiswap/IMasterChef.sol	a1aaad33efdee6f2b78eb2709b7576c738e7d759d237cbc9aaeb3d6b00ba8795
IMV	external/sushiswap/IMasterChefV2.sol	9af52b11e9d7f2231a643fdadbb13e7a3153bcb5a494977309177379438bdedb
UVL	external/sushiswap/UniswapV2Library.sol	57f3c208344eea3dbd7aed6f93340989e6dd0847596ffa0052a100834cefc88c
UVC	external/uniswap/UniswapV2Library.sol	85fc9dd6e465e3bd1eda5cbac834eeca07d343c43652c3e29fdcf275cfb9176b
IWE	external/IWETH.sol	d5f45bad700403c0c5a440fcc57cdf349e3263c55cd725a6f3dd758dc5361091
CRC	refs/CoreReference.sol	7062312b924b6d574b8e2cf345f9fbf1e03f15a70fd0f9623c7019ea8973f813
MCV	vaults/sushiswap/MasterChefV2Vault.sol	9380621fdbf9e9471a8ec729a25d3354d729b169a1899240bea0c8c8e36bc9ee
MCC	vaults/sushiswap/MasterChefVault.sol	83bf3e19dd5aed108c88a3e5ef581ee8866c9cc453a80bd5264867548d07f2c6
SVC	vaults/sushiswap/SushiswapVault.sol	15254eb21858b42b68743aa8470661225461eb7f4412ee8e82238acb06028e6d
SVS	vaults/sushiswap/SushiswapVaultStorage.sol	5377e368a4c48a866688c89b50c184cb73fe89fde32cf5859f5f59ed541be29a
UVK	vaults/uniswap/UniswapVault.sol	f3696933894ee7204c5966bb9a82a856e1b3cdaab2e20f9ae708c587a5e215b8

ID	File	SHA256 Checksum
UVS	vaults/uniswap/UniswapVaultStorage.sol	eacd98fe27c1133de50039aca01318c1180977126a0261d992f6e75982ebf3be
IVC	vaults/IVault.sol	14c57466e893210edf7e293584c39b5c9fc28a0f0f778810d43eed056565adab
VCK	vaults/Vault.sol	b18a9df641f3d698486ebe43a5fe12fdd179c2d1ae1961f2e9394055f940dbe1
VSC	vaults/VaultStorage.sol	f3fb977f0c17de1211d1f1c2b6b19c027634e77d44ba1f02c8a566e988a6ce5f
CCP	core/Core.sol	78b8ac6538d09f4d4acd6a12462a8f781aed2b5802b293317bef150a9521ae0e
CPK	core/CorePermissions.sol	2437720f13ef8b18bf885b56475af6ab150760924b370b621d709539ff4de90a
CSK	core/CoreStorage.sol	fc113c487d56f1526493f5ef141938ad742f883cfce2f50ed1e13bad4eac7726
ICK	core/ICore.sol	778441917442099114dac8b9bb937e9c39450e0e27bfc2294dcb25dbe7240abf
IPC	core/ICorePermissions.sol	b6e03d557a01caa2f0709ff6d2a997546440b67c727a900779a8595a3f8d7196
IMK	external/sushiswap/IMasterChef.sol	a1aaad33efdee6f2b78eb2709b7576c738e7d759d237cbc9aaeb3d6b00ba8795
IMP	external/sushiswap/IMasterChefV2.sol	9af52b11e9d7f2231a643fdadbb13e7a3153bcb5a494977309177379438bdedb
UVP	external/uniswap/UniswapV2Library.sol	85fc9dd6e465e3bd1eda5cbac834eeca07d343c43652c3e29fdcf275cfb9176b
IWC	external/IWrappy.sol	aed4a1cc589a128182137775d241b7d249366e1356140882ad7a6b37f9bd9179
CRK	refs/CoreReference.sol	dc85048b2df04cc75af47dc9f1cef2422c46142d6304e188449b315979188317
MCK	vaults/sushiswap/MasterChefV2Vault.sol	a357adaa03f18986f2c851c92819903e9f7047b1d62197d8693de2e9857a8938
MCP	vaults/sushiswap/MasterChefVault.sol	9a8205fa7bb74ca1a8807ce09d9b17a150de1e844d2dce11b1814cd63d27e028
SVK	vaults/sushiswap/SushiswapVaultStorage.sol	fb5d7de0de7747418adbe2b4d67857f86469b9771463373b81c070d49d631341

ID	File	SHA256 Checksum
UCK	vaults/uniswap/UniswapVault.sol	47bb6b09ca0e4d7790943567846e65b5a3a13f2b8c84ec62651d5d9e4bf9c2d6
USC	vaults/uniswap/UniswapVaultStorage.sol	64663c96e1411d023849cbb602dfca9242caa193d2762f3b46fd4a7b8c74927c
IVK	vaults/IVault.sol	16f792e48ffbeaa46584a2463b4f992dfb2ade991b7e5ae6cbddd4ca0c0ba494
VCP	vaults/Vault.sol	fd0ec14c26332a2d784a45c58c8eafa7ae8f82e8f34ac148ba9c4b05e5bfbdca
VSK	vaults/VaultStorage.sol	31a26239107a617c7e0150988cb6fdb2c2031ba949ab36a70967f5ff452ea209

Overview

The **Rift protocol** implements a novel mechanism to allocate token pair liquidity via epoch-wise swapping such that one side of the pair always gets back their floor position and the other side gets back at most their ceiling position. In this way, it enables deepened liquidity for DAOs by allowing them to forgo the swap fees of an LP position and incentivize the other half of the pair with those returns. The **Rift protocol** consists of vaults, each accepting deposits for two assets, that is, an ERC20 (or native token)-ERC20 pair. A DAO that wants to use the Rift protocol will deposit their DAO tokens into their vault. When another user wants to deposit the token paired with that DAO token, their deposit is paired up with an equivalent amount of DAO tokens and they are deposited into a DEX pool to start earning yield.

The `Core` contract stores all important parameters for the protocol. The main logic of the protocol is implemented in the `vaults` folder that allows the user to deposit, withdraw and keep track of the accounting of their returns. All the DEX type's contracts inherit from the base contract `Vault` which handles the interaction between users and the protocol, including deposits, withdraws and claims.

External Dependencies

The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

There are a few dependent injection contracts or addresses in the current project:

- `AccessControlEnumerableUpgradeable` for the contract `CorePermissions`;
- `EnumerableSetUpgradeable` for the contract `CoreStorage`;
- `IAccessControlUpgradeable` for the contract `ICorePermissions`;
- `IERC20` for the contract `IMasterChef`;
- `IUniswapV2Pair`, `SafeMath` for the contract `UniswapV2Library`;
- `Initializable`, `core` for the contract `CoreReference`;
- `lpToken` for the contract `MasterChefV2Vault` and `MasterChefVault`;
- `SafeERC20Upgradeable` for the contract `MasterChefV2Vault`, `MasterChefVault` and `SushiswapVault`;
- `token0`, `token1` for the contract `UpgradedVault` and `Vault`;
- `IERC20Upgradeable` for the contract `SushiswapVaultStorage`;
- `SafeERC20Upgradeable`, `IUniswapV2Router02` for the contract `UniswapVault`;
- `SafeERC20Upgradeable`, `ReentrancyGuardUpgradeable` for the contract `Vault`;
- `IERC20Upgradeable` for the contract `VaultStorage`;
- `rewarder`, `sushi` for the contract `MasterChefVault` and `MasterChefV2Vault`;
- `factory`, `router` for the contract `UniswapVault` and `SushiswapVault`.

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

Privileged Functions

The contract `CorePermissions` contains the following privileged functions that are restricted by the `governor`:

- `CorePermissions.createRole(bytes32 role, bytes32 adminRole)`: set the `adminRole` as `role`'s admin role;
- `CorePermissions.whitelistAll(address[] memory addresses)`: whitelist addresses;
- `CorePermissions.disableWhitelist()`: disable the whitelist;
- `CorePermissions.enableWhitelist()`: enable the whitelist.

The contract `CorePermissions` contains the following privileged function that is restricted by the `admin`:

- `CorePermissions.revokeRole(bytes32 role, address account)`: revoke the role of an account that has `admin` as the admin role.

The contract `Core` contains the following privileged functions that are restricted by the `governor` role:

- `Core.registerVaults(address[] memory vaults)`: register new vault contracts with `Core`;
- `Core.removeVaults(address[] memory vaults)`: remove vault contracts from `Core`;
- `Core.setProtocolFee(uint256 _protocolFee)`: set new protocolFee;
- `Core.setFeeTo(address _feeTo)`: set new feeTo.

The contract `Core` contains the following privileged functions that are restricted by the `pauser` role:

- `Core.pause()`: pause the Rift protocol;
- `Core.unpause()`: unpause the Rift protocol.

The contract `Vault` contains the following privileged functions that are restricted by the `strategist` role:

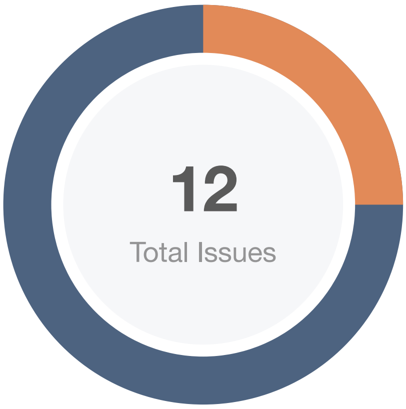
- `Vault.nextEpoch(uint256 expectedPoolToken0, uint256 expectedPoolToken1)`: update the current epoch and move on to the next epoch. Some tokens swap will be performed during this phase, the `strategist` will be responsible of the slippage;
- `Vault.setToken0Floor(uint256 _token0FloorNum)`: set the floor value of `Token0`;
- `Vault.setToken1Floor(uint256 _token1FloorNum)`: set the floor value of `Token1`.

The contract `Vault` contains the following privileged functions that are restricted by the `guardian` role:

- `Vault.rescueTokens(address[] calldata tokens, uint256[] calldata amounts)`: **withdraw funds from this contract to the guardian**;
- `Vault.unstakeLiquidity()`: **unstake any liquidity before rescuing LP tokens**.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings



Critical	**0** (0.00%)
Major	**3** (25.00%)
Medium	**0** (0.00%)
Minor	**0** (0.00%)
Informational	**9** (75.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
CCK-01	Protocol Fee Has Never Been Applied	Logical Issue	Informational	Resolved
CCP-01	Centralization Related Risks in Contract `Core`	**Centralization / Privilege**	**Major**	Acknowledged
CPK-01	Centralization Related Risks in Contract `CorePermissions`	**Centralization / Privilege**	**Major**	Acknowledged
UVK-01	Inconsistency With Protocol Description	Logical Issue	Informational	Resolved
VCK-01	Potential Sandwich Attacks	Logical Issue	Informational	Resolved
VCK-02	Same Local Variable Name	Coding Style	Informational	Resolved
VCK-03	Inconsistent Lines of Code Order	Coding Style	Informational	Resolved
VCK-04	Typos in Comments	Coding Style	Informational	Resolved
VCK-05	Incompatibility With Deflationary Tokens	Volatile Code	Informational	Acknowledged
VCK-06	Gas Optimization on `pendingDeposits` Calculation	Gas Optimization	Informational	Resolved
VCP-01	Centralization Related Risks in Contract `Vault`	**Centralization / Privilege**	**Major**	Acknowledged

GLOBAL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Global	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

The compiler version is recommended to be locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.0` the contract should contain the following line:

```
pragma solidity 0.8.0;
```

Alleviation

[**Rift Finance**]: The team heeded the advice and resolve this issue in the commit a7140c7c5689819c28fca76d1c23c84ab9cd6454.

CCK-01 | Protocol Fee Has Never Been Applied

Category	Severity	Location	Status
Logical Issue	Informational	core/Core.sol (753cdb4): 63	Resolved

Description

The protocol fee logic is defined in contract `Core`, but has never been applied in the vaults.

Recommendation

Recommend either implementing the missing logic related to the protocol fee or removing the unused logic.

Alleviation

[**Rift Finance**]: The protocol fee has been applied in contract `Vault` as follows:

```
        // collect protocol fee if profitable
        if (_token0.available > _token0.original) {
            _token0.available -= ((_token0.available - _token0.original) * core.protocolFee()) / core.MAX_FEE();
        }
        if (_token1.available > _token1.original) {
            _token1.available -= ((_token1.available - _token1.original) * core.protocolFee()) / core.MAX_FEE();
        }
```

The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

CCP-01 | Centralization Related Risks In Contract `Core`

Category	Severity	Location	Status
Centralization / Privilege	Major	core/Core.sol (1ab5422): 149, 142, 101, 109, 90, 78	Acknowledged

Description

In contract `Core`, the role `governor` has the authority over the following functions:

- `Core.registerVaults(address[] memory vaults)`: register new vault contracts with `Core`;
- `Core.removeVaults(address[] memory vaults)`: remove vault contracts from `Core`;
- `Core.setProtocolFee(uint256 _protocolFee)`: set new protocolFee;
- `Core.setFeeTo(address _feeTo)`: set new feeTo.

The role `pauser` has the authority over the following functions:

- `Core.pause()`: pause the Rift protocol;
- `Core.unpause()`: unpause the Rift protocol.

Any compromise to any account with the privileged role may allow the hacker to take advantage of this and disrupt operations involving this contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

[**Rift Finance**]: The team has taken care to delegate responsibilities to independent roles, such as Pauser, Guardian, Strategist, Governor, so that each responsibility is siloed. For example, rescuing funds can only be done by the Guardian, but in order to execute this, the Pauser must have paused the contracts.

Only the Governor can set parameters like the protocol fee. Only the strategist can move the vault to the next epoch, etc. Each of these roles will be managed independently so that a single party does not have control over protocol-wide operations. Additionally, the Governor role(which is the only role with the ability to add roles) will be managed by a multisig.

CPK-01 | Centralization Related Risks In Contract `CorePermissions`

Category	Severity	Location	Status
Centralization / Privilege	Major	core/CorePermissions.sol (1ab5422): 90, 83, 74, 68, 62	Acknowledged

Description

In contract `CorePermissions`, the role `governor` has the authority over the following functions:

- `CorePermissions.createRole(bytes32 role, bytes32 adminRole)`: set the `adminRole` as `role`'s admin role;
- `CorePermissions.whitelistAll(address[] memory addresses)`: whitelist addresses;
- `CorePermissions.disableWhitelist()`: disable the whitelist;
- `CorePermissions.enableWhitelist()`: enable the whitelist.

The role `admin` has the authority over the following function:

- `CorePermissions.revokeRole(bytes32 role, address account)`: revoke the role of an account that has `admin` as the admin role.

Any compromise to any account with the privileged role may allow the hacker to take advantage of this and disrupt operations involving this contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

[**Rift Finance**]: The team has taken care to delegate responsibilities to independent roles, such as Pauser, Guardian, Strategist, Governor, so that each responsibility is siloed. For example, rescuing funds can only be done by the Guardian, but in order to execute this, the Pauser must have paused the contracts.

Only the Governor can set parameters like the protocol fee. Only the strategist can move the vault to the next epoch, etc. Each of these roles will be managed independently so that a single party does not have control over protocol-wide operations. Additionally, the Governor role(which is the only role with the ability to add roles) will be managed by a multisig.

UVK-01 | Inconsistency With Protocol Description

Category	Severity	Location	Status
Logical Issue	Informational	vaults/uniswap/UniswapVault.sol (753cdb4): 3	Resolved

Description

According to the protocol description at Github, the `Vault` contract should include both `token0FloorNum` and `token1FloorNum`.

- "token0FloorNum: interest rate floor for the token0 side, out of 10000. This will be set to ~10000, to guarantee lossless returns for the token0 side."
- "token1FloorNum: interest rate floor for the token1 side, out of 10000. This will be set to a low amount, just to keep internal accounting consistent."

However, per the code implementation, it does not have logic related to `token1FloorNum` mentioned in the protocol description.

Recommendation

The auditing team would like to know if it is the intended design or the team plan to add `token1FloorNum` logic in the later stage.

Alleviation

[**Rift Finance**]: The team heeded the advice and resolved the issue by implementing the logic related to `token1FloorNum`. For example,

```
        uint256 token0Floor = _token0Data.reserves + (_token0Data.active * token0FloorNum) / DENOM;
        uint256 token1Floor = _token1Data.reserves + (_token1Data.active * token1FloorNum) / DENOM;
```

The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

VCK-01 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	● Informational	vaults/Vault.sol (753cdb4): 414~418	✓ Resolved

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back-running (after the transaction being attacked) a transaction to sell the asset.

In order to protect itself against sandwich attacks, the project implements the `_nextEpoch()` function to ensure that the pools have a desired amount of tokens before proceeding to the swap, the corresponding variables are: `expectedPoolToken0` and `expectedPoolToken1`.

The `expectedPoolToken0` and `expectedPoolToken1` are chosen by the `Governor`, who is the one allowed to call `nextEpoch()`.

```
function nextEpoch(uint256 expectedPoolToken0, uint256 expectedPoolToken1)
    public
    override
    onlyGovernor
    whenNotPaused
```

This means that in case of a human error from the `Governor`, sandwich attacks could still occur.

Recommendation

The auditing team would like to understand how the `nextEpoch()` function will be called in order to limit the risks of errors. In particular, how to calculate the values of `expectedPoolToken0` and `expectedPoolToken1` that are passed to `nextEpoch()`.

Alleviation

[**Rift Finance**]: This will be called via "manager dashboard" that allows the governor to view the current balances(like the Uniswap UI does in the background) so that valid amounts can be used. Sandwich attacks are impossible if arguments are passed correctly.

[**CertiK**]: The Rift Finance team will ensure the correctness of the input parameters.

VCK-02 | Same Local Variable Name

Category	Severity	Location	Status
Coding Style	Informational	vaults/Vault.sol (753cdb4): 438, 462	Resolved

Description

The aforementioned variable `neededToSwap` has been used to represent both the amount of token0 and token1 required to obtain a certain amount of token1 and token0, respectively.

Recommendation

To improve code readability, recommend using different local variable names to distinguish them.

Alleviation

[**Rift Finance**]: The team heeded the advice and resolved the issue by using `token1NeededToSwap` and `token0NeededToSwap` to distinguish the `neededToSwap` amount of token1 from token0. The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

VCK-03 | Inconsistent Lines Of Code Order

Category	Severity	Location	Status
Coding Style	● Informational	vaults/Vault.sol (753cdb4): 476~477	Resolved

Description

To be consistent with the coding style in previous lines (#L471-472), the aforementioned two lines of code (#L476-477) are supposed to exchange location.

For example,

```
471            _token0.available -= amountConsumed;
472            _token1.available += amountOut;
```

```
476            _token1.available += amountOut;
477            _token0.available -= amountConsumed;
```

Recommendation

Recommend exchanging the aforementioned two lines of code for code consistency.

Example,

```
476            _token0.available -= amountConsumed;
477            _token1.available += amountOut;
```

Alleviation

[**Rift Finance**]: The team heeded the advice and resolved the issue by exchanging the two lines of code as follows:

```
471            _token0.available -= amountConsumed;
472            _token1.available += amountOut;
```

The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

VCK-04 | Typos In Comments

Category	Severity	Location	Status
Coding Style	Informational	vaults/Vault.sol (753cdb4): 234, 213, 227, 544	Resolved

Description

There are several typos in the comments.

1. `deposit requests are processed` should be `withdraw requests are processed`.
2. `Withdraws all liquidity` should be `Deposits all liquidity`.

Comments are programmer-readable explanations or annotations in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand. Inaccurate comments may mislead readers.

Recommendation

Recommend fixing the typos for better readability.

Alleviation

[**Rift Finance**]: The team heeded the advice and resolved the issue by correcting the aforementioned typo in the comments. The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

VCK-05 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	Informational	vaults/Vault.sol (753cdb4): 59~60	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and might cause some exceptions.

For example,

```
90    token1.safeTransferFrom(msg.sender, address(this), _amount);
```

If `token1` is a deflationary token, the deposited amount of `token1` in the protocol would be less than the initial amount the user transferred.

Recommendation

Recommend regulating the set of token pairs supported in the Rift protocol, and, if there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

Alleviation

[**Rift Finance**]: The team updated the documentation to indicate that the protocol does not support deflationary tokens.

VCK-06 | Gas Optimization On `pendingDeposits` Calculation

Category	Severity	Location	Status
Gas Optimization	● Informational	vaults/Vault.sol (753cdb4): 330	Resolved

Description

The `pendingDeposits` is the amount of the pending deposit of some user in current epoch. It is initialized in Line 316 with a default value of 0. If the current epoch (i.e., `currEpoch`) is same to the deposit epoch (i.e., `depositEpoch`), it should be the previously recorded deposit value (i.e., `pendingDeposits`).

```
326             if (depositEpoch < currEpoch) {
327                 balanceDay0 += (depositAmt * RAY) /
assetData.epochToRate[depositEpoch];
328             } else {
329                 // if they have one from this epoch, add the flat amount
330                 pendingDeposits += depositAmt;
331             }
```

However, in Line 330, when calculating `pendingDeposits`, it adds `depositAmt` instead of directly assigning the value of `depositAmt` to `pendingDeposits`, which leads to extra gas cost.

Recommendation

Recommended modifying the code in L330 to

```
330                 pendingDeposits = depositAmt;
```

Alleviation

[**Rift Finance**]: The team heeded the advice and resolved the issue by making the recommended change. The changes are reflected in the commit `1ab54221a7266331a6fa17cb1d9da5d028a14814`.

VCP-01 | Centralization Related Risks In Contract `Vault`

Category	Severity	Location	Status
Centralization / Privilege	Major	vaults/Vault.sol (1ab5422): 603, 579, 701, 693, 384	Acknowledged

Description

In contract `Vault`, the role `strategist` has the authority over the following functions:

- `Vault.nextEpoch(uint256 expectedPoolToken0, uint256 expectedPoolToken1)`: update the current epoch and move on to the next epoch. Some tokens swap will be performed during this phase, the `strategist` will be responsible of the slippage;
- `Vault.setToken0Floor(uint256 _token0FloorNum)`: set the floor value of `Token0`;
- `Vault.setToken1Floor(uint256 _token1FloorNum)`: set the floor value of `Token1`.

The role `guardian` has the authority over the following functions:

- `Vault.rescueTokens(address[] calldata tokens, uint256[] calldata amounts)`: **withdraw funds from this contract to the guardian**;
- `Vault.unstakeLiquidity()`: **unstake any liquidity before rescuing LP tokens**.

Any compromise to any account with the privileged roles may allow the hacker to take advantage of this and disrupt operations involving this contract.
Especially, if a `guardian` account got compromised, funds of the entire vault could be stolen. Considering the `token0` and `token1` are from users' deposits, it could lead to the loss of users' funds.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

[**Rift Finance**]: The team has taken care to delegate responsibilities to independent roles, such as Pauser, Guardian, Strategist, Governor, so that each responsibility is siloed. For example, rescuing funds can only be done by the Guardian, but in order to execute this, the Pauser must have paused the contracts.

Only the Governor can set parameters like the protocol fee. Only the strategist can move the vault to the next epoch, etc. Each of these roles will be managed independently so that a single party does not have control over protocol-wide operations. Additionally, the Governor role(which is the only role with the ability to add roles) will be managed by a multisig.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK