Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Range Protocol

AMM Vault

Veridise Inc.
April 3, 2024

► **Prepared For:**

Range Protocol
`https://www.rangeprotocol.com`

► **Prepared By:**

Daniel Dominguez
Alberto Gonzalez

► **Contact Us:** `contact@veridise.com`

► **Version History:**

March. 29, 2024. Initial draft
April. 3, 2024. Official report

Contents

1 Executive Summary

From Mar. 14, 2024 to Mar. 22, 2024, Range Protocol engaged Veridise to review the security of their AMM Vaults. The review covered the AMM Vaults for the Uniswap, IZumi, Algebra, and Pancakeswap DEXs. Specifically, the review encompassed the conversion of the Vault main logic to a library, additional rebalance logic, and the implementation of a new type of fee. It is important to note that while Veridise audited the previous version of the Uniswap Vault*, the audits for the other DEXs were not conducted.

Veridise conducted the assessment over 2 person-weeks, with 2 engineers reviewing code over 1 weeks on the following diffs:

- Uniswap: `27b2b08->8b8347`
- IZumi: `651d30c->2dcee64`
- Algebra: `628a1b4->bab9539`
- Pancakeswap: `5be0e44->dfad04e`
- Pancakeswap-stader: `dfad04e->904b462`

The auditing strategy involved extensive manual auditing performed by Veridise engineers.

Code assessment. The AMM Vault developers provided the source code of the AMM Vault contracts for examination. To assist the Veridise auditors' in comprehending the code the developers met with the Veridise team to give a walk-through of the code and point out areas of potential concern.

The source code included a test suite, which the Veridise auditors noted tested both positive and negative paths, verifying most of the access-control related paths as well as a good coverage of different user-flow scenarios.

The Veridise auditors found the code to be well-structured and adhering to Solidity best practices. It is worth emphasizing the code's clarity and organized structure, which allowed the auditors to focus on its security aspects.

Summary of issues detected. The audit uncovered 11 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. Among these, 3 medium-severity issues were identified. One of these, V-RMV-VUL-002, pointed out an error in the computation of the vault's passive balance, as it failed to include the last collected fees. Another medium severity issue, V-RMV-VUL-003, highlighted a potential leftover of native tokens during the burn function due to an incorrect transfer amount. The Veridise auditors identified 1 low-severity issues. Specifically, V-RMV-VUL-004 pointed out to an off-by-one incorrect comparison which can cause an incorrect computation of the vault's passive balance. In addition, the Veridise auditors uncovered 2 warning findings and 5 informational findings.

* The previous audit report can be found on Veridise's website at `https://veridise.com/audits/`

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the AMM Vault security:

- The auditors recommend the Range protocol to be the first depositor in each vault to avoid any risk of share price manipulation.
- The auditors recommend not allowing mint and burn transactions to go through in the same block to avoid any risk of share price manipulation.
- The auditors recommend using a multisig account for the factory owner as it has the access control to upgrade the vault logic. As well as key management standards, for the manager accounts as they are a trusted point in the protocol.

Finally, the Veridise team recommends active monitoring of the protocol as an additional proactive defensive measure.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Uniswap	`27b2b08->8b8347`	Solidity	EVM
IZumi	`651d30c->2dcee64`	Solidity	EVM
Algebra	`628a1b4->bab9539`	Solidity	EVM
Pancakeswap	`5be0e44->dfad04e`	Solidity	EVM
Pancakeswap-stader	`dfad04e->904b462`	Solidity	EVM

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 14 - Mar. 22, 2024	Manual	2	2 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	3	2	2
Low-Severity Issues	1	1	1
Warning-Severity Issues	2	0	2
Informational-Severity Issues	5	4	4
TOTAL	11	7	9

Table 2.4: Category Breakdown.

Name	Number
Data Validation	4
Logic Error	3
Maintainability	2
Access Control	1
Usability Issue	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of diffs between the AMM Vault's smart contracts. In our audit, we sought to answer questions such as:

- Does the conversion into a library changed the semantics of the code?.
- What are the semantic differences across different DEXs vaults?.
- Is the new fee type applied consistently as the other fees?.
- Are external APIs used correctly?
- Can users profit from well-timed minting and burning during a rebalance operation?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of our in-house static analyzer, Vanguard.

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit was limited to the semantic differences between the code diffs outlined in the Table 2.1. The Veridise auditors assumed the previous logic was correctly implemented.

Methodology. Veridise auditors reviewed the reports of previous audits for AMM Vault, inspected the provided tests, and read the AMM Vault documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing. Previous to the audit, the Veridise auditors met with the AMM Vault developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RMV-VUL-001	The burn function lacks the whenNotPaused modi	Medium	Intended Behavior
V-RMV-VUL-002	The burn function is not subtracting the recent. . .	Medium	Fixed
V-RMV-VUL-003	Incorrect withdrawal amount for native tokens	Medium	Fixed
V-RMV-VUL-004	Incorrect computation of underlying balances	Low	Fixed
V-RMV-VUL-005	The mint function slippage protection lacks min. . .	Warning	Acknowledged
V-RMV-VUL-006	Centralization Risk	Warning	Acknowledged
V-RMV-VUL-007	Performance and other fee can be set incorrectly	Info	Fixed
V-RMV-VUL-008	Chainlink feeds are not consumed with best prac. .	Info	Fixed
V-RMV-VUL-009	The swap function does not have exactOutput sli. .	Info	Intended Behavior
V-RMV-VUL-010	Typos and incorrect comments	Info	Fixed
V-RMV-VUL-011	Differences between vault implementations	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-RMV-VUL-001: The burn function lacks the whenNotPaused modifier

Severity	Medium	**Commit**	See issue description
Type	Data Validation	**Status**	Intended Behavior
File(s)	`RangeProtocolVault.sol`		
Location(s)	burn()		
Confirmed Fix At	N/A		

The pause mechanism is implemented to halt the minting and burning features of the Vault in response to any security risks. However, there is an issue with the burn function: it lacks the `whenNotPaused` modifier that was present in the previous version of the function.

Impact As a result, the emergency pause mechanism will not have affect over the burn function.

This issue is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.
- The Pancakeswap-stader vault at commit: `904b462`.

Recommendation To address this issue, add the `whenNotPaused` modifier to the burn function.

Developer Response To have the burn function not pausable is the intended behaviour. Our rationale is that users should be able to close their positions at any time.

4.1.2 V-RMV-VUL-002: The burn function is not subtracting the recently collected fees

Severity	Medium	**Commit**	See issue description
Type	Logic Error	**Status**	Fixed
File(s)	`VaultLib.sol`		
Location(s)	burn()		
Confirmed Fix At	de02b78, 80a59f9, f910db7, 2db602d		

During the `burn` function, the code calculates the amount of passive balance without including the manager and other fees:

```
if (vars.passiveBalance0 > vars.feeBalance0) {
     vars.passiveBalance0 -= vars.feeBalance0;
}

if (vars.passiveBalance1 > vars.feeBalance1) {
   vars.passiveBalance1 -= vars.feeBalance1;
}
```

Snippet 4.1: Code snippet from the `burn` function. It subtracts `feeBalance` from `passiveBalance`.

The `feeBalance` variable is initialized at the beginning of the function as the sum of `managerBalance` + `otherBalance`. However, the `feeBalance` does not account for the increase in the manager and other balances that occur during `_applyPerformanceAndOtherFee`.

The previous version of the code did not have this issue because it directly used the state variable `managerBalance` instead of a previously cached local version of it.

Impact As a result, the code will incorrectly return a portion of the recently collected fees to the user, which should remain in the contract's balance.

This issue is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.
- The Pancakeswap-stader vault at commit: `904b462`.

Recommendation To address this, consider the recent increase in the manager and other fees when subtracting `feeBalance` from `passiveBalance`.

Developer Response applied the fixes.

4.1.3 V-RMV-VUL-003: Incorrect withdrawal amount for native tokens

Severity	Medium	**Commit**	See issue description
Type	Logic Error	**Status**	Fixed
File(s)	LibVault.sol		
Location(s)	_processWithdraw()		
Confirmed Fix At	2db602d, 0db598f		

The Pancakeswap based versions of the Vault include a feature to handle native token deposits and withdrawals during minting and burning. The burn function manages this through the `_processWithdraw` internal function, which transfers the native token to the sender at the end of the function:

```
if (amount0 != 0) {
    if (isToken0Native && withdrawNative) {
           weth9Amount = amount0;
    } else {
           token0.safeTransfer(msg.sender, amount0);
    }
}

if (amount1 != 0) {
     if (isToken1Native && withdrawNative) {
            weth9Amount = amount1;
     } else {
            token1.safeTransfer(msg.sender, amount1);
     }
}

if (weth9Amount != 0) {
    WETH9.withdraw(amount0);
    msg.sender.call{value: weth9Amount}("");
}
```

Snippet 4.2: Code snippet from the `_processWithdraw` function.

However, the amount withdrawn from the wrapped contract is `amount0` instead of `weth9Amount`, which is incorrect when `token0 != WETH9`. It is important to note that the previous version of the Vault does not have this issue.

Impact This issue can have two possible impacts:

1. If `amount0 < weth9Amount`, the mechanism of native tokens withdrawals will enter a denial of service state, as it will attempt to send more native tokens than available in its balance.
2. If `amount0 > weth9Amount`, the contract will be left with some native tokens that will be credited to the next minter. This occurs because the following line at the end of the mint function refunds the native tokens to the sender:

```
if (address(this).balance != 0) msg.sender.call{value: address(this).balance}("");
```

Snippet 4.3: Code snippet from `mint` function.

This issue is present in the following instances:

- The Pancakeswap-stader vault at commit: `904b462`.
- The Pancakeswap vault at commit: `dfad04e`

Recommendation To address this issue, it is recommended to change `amount0` to `weth9Amount` in the withdrawal process to ensure that the correct amount of wrapped tokens is withdrawn from the wrapped contract.

Developer Response applied the fixes.

4.1.4 V-RMV-VUL-004: Incorrect computation of underlying balances

Severity	Low	**Commit**	See issue description
Type	Logic Error	**Status**	Fixed
File(s)	`VaultLib.sol`		
Location(s)	_getUnderlyingBalances()		
Confirmed Fix At	de02b78, 80a59f9, f910db7, 2db602d		

When computing the underlying `token0` and `token1` balances of the contract it is important to consider the passive balance of the contract, that is, balance that is in the current balance of the contract for both tokens. But it is important to consider that `managerBalance` and `otherBalance` may be part of the passive balance of the contract, this case is considered in the code of the Vault:

```
vars.passiveBalance0 = state.token0.balanceOf(address(this));
vars.passiveBalance1 = state.token1.balanceOf(address(this));

vars.feeBalance0 = state.managerBalance0 + state.otherBalance0;
vars.feeBalance1 = state.managerBalance1 + state.otherBalance1;

// @audit-issue Should be >= instead of >.
amount0Current += vars.passiveBalance0 > vars.feeBalance0
      ? vars.passiveBalance0 - vars.feeBalance0
      : vars.passiveBalance0;

amount1Current += vars.passiveBalance1 > vars.feeBalance1
      ? vars.passiveBalance1 - vars.feeBalance1
      : vars.passiveBalance1;
```

Snippet 4.4: Code snippet from the `_getUnderlyingBalances` function.

However, the code has a flaw when `passiveBalance == feeBalance`. The code will not subtract the fee balance making the vault to think that it hold more tokens that it really does.

Impact The vault will return incorrect underlying balances if `passiveBalance == feeBalance`. The same issue is present in the `burn` function.

This issue (in both functions) is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.
- The Pancakeswap-stader vault at commit: `904b462`.

Recommendation To address this issue, change `> to >=`.

Developer Response applied the fixes.

4.1.5 V-RMV-VUL-005: The mint function slippage protection lacks minimum amount validation

Severity	Warning	**Commit**	See issue description
Type	Data Validation	**Status**	Acknowledged
File(s)	`VaultLib.sol`		
Location(s)	mint()		
Confirmed Fix At	N/A		

When adding liquidity, it's crucial to verify the price at which liquidity is being added. Therefore, slippage protection should not only specify the maximum amount of tokens but also the minimum amount of tokens to supply. However, the `mint` function currently only verifies the maximum amount of tokens and not the minimum amount of tokens that were added.

Impact This oversight allows malicious users to sandwich the Vault's mint operations, causing them to add liquidity at an incorrect price. For example, if `tokenA` and `tokenB` have a current price of `1:1`, a user could specify a maximum amount of `1000` tokens for each token. However, a transaction that adds `800` `tokenA` and `1000` `tokenB` would still succeed.

This issue is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.
- The Pancakeswap-stader vault at commit: `904b462`.

Recommendation Follow the same pattern found in `addLiquidity`:

```
if (
    amountDeposited0 < minAmountsIn[0] ||
    amountDeposited0 > maxAmountsIn[0] ||
    amountDeposited1 < minAmountsIn[1] ||
    amountDeposited1 > maxAmountsIn[1]
) revert VaultErrors.SlippageExceedThreshold();
```

Snippet 4.5: Code snippet from the `addLiquidity` function. It checks for maximum and minimum deposited amounts.

Developer Response When the price moves one of the token will increase in amount and the other decrease so the transaction will still fail. That was the reason to omit the lower bound check.

4.1.6 V-RMV-VUL-006: Centralization Risk

Severity	Warning	**Commit**	type commit hash here
Type	Access Control	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

The Vaults designates a manager account with special permissions, including:

- The ability to pause the contract.
- Re-allocate the liquidity in a different range
- Increment the fees, even to 100% in the case of the `otherFee`.
- Make the vault perform arbitrary calls via the `rebalance` function.

It is important to note that the Factory's owner has the special permission to be able to upgrade the Vault logic completely.

Impact If the private key of the manager or the owner (of the factory) accounts are stolen, the attacker can drain the vault's fund via an upgrade or disrupt the operations of the Vault.

Recommendation We recommend the factory owner to be behind a multisig or timelock and key management standards for the manager account.

Developer Response All the owners of Factories are Timelock contracts and all the Timelock contracts are controlled by Multisigs.

4.1.7 V-RMV-VUL-007: Performance and other fee can be set incorrectly

Severity	Info	**Commit**	See issue description
Type	Data Validation	**Status**	Fixed
File(s)	`VaultLib.sol`		
Location(s)	_updateFees()		
Confirmed Fix At	fe55099, 80a59f9, f910db7, 2db602d		

Two types of fees are deducted from the swap fees earned by the Vault position: the performance fee and the "other" fee. An implicit invariant is that the sum of these fees should not exceed 100%. However, this validation is missing when updating the fees.

Impact It is possible to set the fees for their sum to be greater than the 100%.

This issue is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.

Recommendation Check that `newOtherFee + performanceFee <= 10_000` when updating them.

Developer Response applied the fixes.

4.1.8 V-RMV-VUL-008: Chainlink feeds are not consumed with best practices

Severity	Info	**Commit**	See issue description
Type	Data Validation	**Status**	Fixed
File(s)	`VaultLib.sol`		
Location(s)	rebalance()		
Confirmed Fix At	fe55099, 02f5fb3, 10130e4, b406dd1		

When consuming Chainlink price feeds, it is crucial to perform sanity checks and staleness checks on the returned price. However, the current code in the `rebalance` function lacks these validations.

References:

- Chainlink Aggregator Contract on Etherscan - Call the `latestRoundData` function.
- How to Consume Chainlink Price Feeds Safely
- Chainlink Official Docs

Impact The vault might consume an invalid or outdated price. This issue is present in the following instances:

- The Uniswap vault at commit: `8b83479`.
- The IZumi vault at commit: `2dcee64`.
- The Algebra-based vault at commit: `bab9539`.
- The Pancakeswap vault at commit: `dfad04e`.

Recommendation To mitigate this risk, it is recommended to implement the sanity and staleness checks as recommended by Chainlink. These checks can help ensure that the price data used by the vault is reliable and up-to-date, reducing the likelihood of incorrect decisions based on inaccurate price information.

Developer Response applied the fixes with the last update of price oracle being less than heartbeat.

4.1.9 V-RMV-VUL-009: The swap function does not have exactOutput slippage protection

Severity	Info	**Commit**	See issue description
Type	Usability Issue	**Status**	Intended Behavior
File(s)	VaultLib.sol		
Location(s)	swap()		
Confirmed Fix At	N/A		

The swap function currently only includes slippage protection for exact input swaps. The provided code ensures that the amount received from the swap is not less than a specified minimum. However, in the case of an exact output swap, it should also verify that the amount sent is not greater than a specified maximum.

```
if (
     (zeroForOne && uint256(-amount1) < minAmountOut) ||
     (!zeroForOne && uint256(-amount0) < minAmountOut)
) revert VaultErrors.SlippageExceedThreshold();
```

Snippet 4.6: Code snippet from the swap function. It only validates the received amounts.

Impact If the manager executes an exact output swap (by passing a negative swapAmount), they will not be protected from making a poor swap.

This issue is present in the following instances:

- The Uniswap vault at commit: 8b83479.
- The Algebra-based vault at commit: bab9539.
- The Pancakeswap vault at commit: dfad04e.
- The Pancakeswap-stader vault at commit: 904b462.

Recommendation To address this issue, add support for slippage protection in exact output swaps. You can refer to the reference implementation at the following link for guidance:

- Uniswap SwapRouter contract

Developer Response Our swap function is intended to support only exact input swap.

4.1.10 V-RMV-VUL-010: Typos and incorrect comments

Severity	Info	**Commit**	See issue description
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description In the following locations, the auditors identified minor typos and potentially misleading comments:

- In the IZumi vault (`2dcee64`) in file `VaultLib.sol`:
  - Line 327: Comment belongs to the function with the same name in the caller contract and does not properly reflect the behavior of the function in `VaultLib.sol`
- In the Pancakeswap vault (`dfad04e`) in file `VaultLib.sol`:
  - Line 450: Should say **maximum** instead of **minimum**

Impact These minor errors may lead to future developer confusion.

Developer Response applied the fixes.

4.1.11 V-RMV-VUL-011: Differences between vault implementations

Severity	Info	**Commit**	See issue description
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	fe55099, 02f5fb3, 3f76c8c, b406dd1, 7ea327b		

Implementation differences have been identified in the following instances:

- **IZumi** vault at commit: `2dcee64`.
- **Algebra**-based vault at commit: `bab9539`.
- **Pancakeswap** vault at commit: `dfad04e`.
- **Uniswap** vault at commit: `8b83479`
- **Pancakeswap-stader** vault at commit: `904b462`.

Differences The `deployer` account in the Factory contract:

- The Factory contracts for the **Uniswap** and **IZumi** vaults include the `deployer` account, which is the only account that can create new vaults. However, this account is not present in the **Algebra** and **Pancakeswap** vaults.

The `otherFeeClaimer` account in the Vault contract:

- The **Uniswap** vault includes an account named `otherFeeClaimer`, which is the only account that can collect the `otherFee`. In contrast, the other vaults lack this account and rely on the manager to perform such action.

The `collectOtherFee` function in the Vault contract:

- The **Uniswap** vault, when collecting the "other fee," retrieves the most up-to-date fees by calling `_pullFreeFromPool`. However, the other vaults do not call this function when collecting the other fee.

The `receive` function in the Vault contract:

- In the **Pancakeswap** vault, the `receive` function reverts every ether transfer unless the sender is the `WETH9` contract. However, the **Pancakeswap-stader** vault allows receiving ether from any sender.

Impact While these findings do not pose a direct security risk, they can lead to inconsistencies and potential confusion for developers.

Recommendation It is recommended to document these differences across the different types of vaults to provide clarity and ensure consistency in how they are used and managed reducing the risk of errors and misunderstandings.

Developer Response applied the fixes. For Factory implementation, any new factory we deploy will have the implementation with the deployer. Have not made Factory implementation uniform yet but that would be the plan.