Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Range Protocol

Vertex Vault

Veridise Inc.
March 27, 2024

► **Prepared For:**

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

► **Prepared By:**

Daniel Dominguez
Alberto Gonzalez

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

► **Version History:**

March. 14, 2024 Initial Draft
March. 25, 2024 Second version Draft
March. 27, 2024 Official Report

Contents

Executive Summary 1

From Mar. 11, 2024 to Mar. 13, 2024, Range Protocol engaged Veridise to review the security of their Vertex Vault. This Vault is designed to store user funds. A manager is responsible for using the Vault funds to engage in perpetual trading on the Vertex platform to produce profit for the Vault depositors in return for fees.

Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commits `341fd93-6864054`. Specifically, The audit started at commit `341fd93`, followed by Range Protocol developers integrating fixes and new logic code into the vault, which was audited at commit `6864054`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Vertex Vault developers provided the source code of the Vertex Vault contracts for review. To facilitate the Veridise auditors' understanding of the code, the developers provided online documentation explaining the high-level intent of the project. Developers also met with the Veridise team to give an in-depth walk-through of the code and point out areas of potential concern. The source code also contained some documentation in the form of comments on functions and storage variables.

The source code contained a test suite, which the Veridise auditors noted tested both positive and negative paths, verifying most of the access-control related paths.

The Veridise auditors felt that the code was well organized and followed Solidity best practices. It is important to emphasize the clarity and well-structured nature of the code, which enabled auditors to focus on its security aspects.

The vault had a very limited interface, which also reduces the attack surface. Note that this is under the assumption that the manager is a fully trusted entity, which was indicated to the auditors by the Range Protocol team.

Summary of issues detected. The audit uncovered 7 issues, 1 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-VTX-VUL-001 identified a denial of service attack over the mint and burn functions of the Vault. In addition, the Veridise auditors identified 2 low-severity issues. Specifically, V-VTX-VUL-002 identified an unsafe typecast that will break the protocol in the case of the black swan event that the vertex positions becomes negative; V-VTX-VUL-003 identified the lack of slippage protection for users when minting. The audit also uncovered 3 warnings as well. In particular, V-VTX-VUL-005 identified centralization risks that can be patched in order to better protect the protocol in case the manager account gets compromised.

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

- The auditors recommend to implement slippage protection as described in the issue V-VTX-VUL-003.
- The auditors also recommend using key management standards, as the manager is a trusted point in the protocol. To reduce the impact of a potential account compromise the codebase should include the recommendations from the issue V-VTX-VUL-005.

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.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Vertex Vault	`341fd93-6864054`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 11 - Mar. 13, 2024	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

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

Table 2.4: Category Breakdown.

Name	Number
Denial of Service	1
Data Validation	1
Transaction Ordering	1
Logic Error	1
Authorization	1
Access Control	1
Maintainability	1

Audit Goals and Scope 3

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Vertex Vault's smart contracts. In our audit, we sought to answer the following questions:

- Are the computations for minting and burning shares done correctly?
- Can users profit from well-timed minting and burning?
- How can frontrunners profit from interactions with the vault?
- Can funds become locked in the vault?
- How can interactions with Vertex can affect the solvency of the vault?
- Is the Vertex's API used correctly?
- How much damage can the manager do to the vault in case of a private key compromised?

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. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy, flash loan attacks, uninitialized variables, and uses of variables before they are defined.

Scope. The scope of this audit is limited to the `src/` folder of the source code provided by the Vertex Vault developers, which contains the smart contract implementation of the Vertex Vault. Specifically:

- `RangeProtocolVertexVault.sol`
- `RangeProtocolVertexVaultStorage.sol`
- `errors/`, `access/` and `interfaces/` directories.

Methodology. Veridise auditors reviewed the reports of previous audits for Vertex Vault, inspected the provided tests, and read the Vertex Vault documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing.

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.

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

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

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

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

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-VTX-VUL-001	The mint and burn functions can be DoSed	High	Fixed
V-VTX-VUL-002	Unsafe typecast for signedBalance	Low	Fixed
V-VTX-VUL-003	Mint function lacks slippage protection	Low	Fixed
V-VTX-VUL-004	Incorrect comparison between passiveBalance and	Warning	Fixed
V-VTX-VUL-005	Centralization Risk	Warning	Fixed
V-VTX-VUL-006	Unsafe management transfer	Warning	Fixed
V-VTX-VUL-007	Missing comments about implementation upgrade	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-VTX-VUL-001: The mint and burn functions can be DoSed

Severity	High	**Commit**	6864054
Type	Denial of Service	**Status**	Fixed
File(s)	`RangeProtocolVertexVault.sol`		
Location(s)	mint(), burn()		
Confirmed Fix At	bb91302		

Both the `mint()` and `burn()` functions contain a check that reverts execution if the Vault has a pending balance. The pending balance represents USDC in transit from the Vault to the Vertex contracts. However, any user can deposit USDC on behalf of the Vault via the `depositCollateralWithReferral` function in the endpoint contract, potentially allowing a malicious user to perform a denial-of-service (DoS) attack by repeatedly depositing small amounts of USDC.

Impact A malicious user could exploit this vulnerability to disrupt the `mint()` and `burn()` functions, potentially causing inconvenience or financial loss.

Recommendation We recommend either removing the check entirely or implementing a minimum threshold for the pending balance to make the attack more expensive. If the second option is chosen, consider adding a grace period for the `burn()` function to allow emergency actions in case of a legitimate need to burn shares without waiting for pending balances to clear.

Developer Response We have removed the getPendingBalance check.

4.1.2 V-VTX-VUL-002: Unsafe typecast for signedBalance

Severity	Low	**Commit**	341fd93
Type	Data Validation	**Status**	Fixed
File(s)	`RangeProtocolVertexVault.sol`		
Location(s)	getUnderlyingBalance()		
Confirmed Fix At	6864054		

The `getUnderlyingBalance` function contains a typecast of the `signedBalance` variable from `int256` to `uint256` without validating that `signedBalance` is greater than zero. While the likelihood of `signedBalance` being negative is low (indicating the Vault has lost all its money on Vertex), it is important to handle this scenario correctly to prevent potential issues.

Impact If the Vault were to have a negative `signedBalance`, the typecast could cause an inflation of the share price of the contract, leading to incorrect calculations and potentially financial losses for users.

Recommendation It is recommended to add a check to verify that `signedBalance` is greater than or equal to zero before performing the typecast. If `signedBalance` is negative, the execution should be reverted. Consider using OpenZeppelin's safecast library for safe typecasting operations.

Developer Response Added the check recommended check. Optimistically, the signed balance should never be less than zero since before that the account should be liquidated on Vertex.

4.1.3 V-VTX-VUL-003: Mint function lacks slippage protection

Severity	Low	**Commit**	341fd93
Type	Transaction Ordering	**Status**	Fixed
File(s)	`RangeProtocolVertexVault.sol`		
Location(s)	mint		
Confirmed Fix At	6864054		

User transactions, such as minting shares in the Vertex Vault, are not guaranteed to be executed at the same blockchain state as when they were submitted. This lack of guaranteed execution state can result in transactions being executed at a higher share price than expected, exposing users to potential undesired state.

Impact Without slippage protection, users are vulnerable to sudden state changes in the Vertex contract or network congestion, which can cause transactions to be executed at undesired share prices.

Recommendation Implement slippage protection in the `mint` function by including a parameter for the minimum amount of shares the user is willing to accept. This parameter should allow users to specify the worst share price they are willing to accept, similar to the `minAmount` parameter in the `burn` function. This will help protect users from unexpected share price fluctuations and ensure they receive shares at a price they are comfortable with.

Developer Response The check is added.

4.1.4 V-VTX-VUL-004: Incorrect comparison between passiveBalance and managerBalance

Severity	Warning	**Commit**	341fd93
Type	Logic Error	**Status**	Fixed
File(s)	`RangeProtocolVertexVault.sol`		
Location(s)	getUnderlyingBalance()		
Confirmed Fix At	6864054		

When computing the underlying balance of the Vault in the `getUnderlyingBalance()` function, it is important to consider the USDC balance of the contract itself (`passiveBalance`). However, part of this passive balance could be from the `managerBalance`, which should be subtracted from the passive balance. The current implementation incorrectly checks if `passiveBalance` is greater than `managerBalance` before subtracting the `managerBalance`, which could lead to an incorrect calculation if `passiveBalance` is equal to `managerBalance`.

Impact When `passiveBalance` is equal to `managerBalance`, the contract will incorrectly think that the `managerBalance` is part of the underlying balance for share price computation.

Recommendation Change the check in the `getUnderlyingBalance()` function from `passiveBalance > managerBalance` to `passiveBalance >= managerBalance`.

Developer Response The check is added.

4.1.5 V-VTX-VUL-005: Centralization Risk

Severity	Warning	**Commit**	341fd93
Type	Authorization	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	6864054		

The Vertex vault designates a manager account with special permissions, including the ability to upgrade the contract, pause and unpause the contract, add and remove products, use multicall, and change and collect the managing fee. However, granting the manager account the ability to upgrade the contract and add/remove products also allows this account to potentially extract funds from the contract, posing a security risk.

Impact If the private key of the manager account is stolen, the attacker can upgrade the vault logic to arbitrary code or remove products, causing the vault to believe shares are worth less USDC and allowing the attacker to mint shares at a discounted price.

Recommendation We recommend the following measures to mitigate these risks:

1. Separate the accounts responsible for managing the vault and upgrading the vault. Use a multisig account for upgrading the vault to increase security. Delegate the management of the vault to a less secure account, reducing the impact of a compromised manager account.
2. Implement a cooldown period for removing products from the vault. This would provide a window of time for the Range team to take action in case the manager account is compromised, reducing the likelihood of immediate fund extraction by an attacker.

These measures can help enhance the security of the Vertex vault and reduce the risk of unauthorized fund extraction due to a compromised manager account.

Developer Response We have added the upgrade role. The role will be assigned to a Timelock contract upon vault upgrade.

4.1.6 V-VTX-VUL-006: Unsafe management transfer

Severity	Warning	**Commit**	341fd93
Type	Access Control	**Status**	Fixed
File(s)	`OwnableUpgradeable.sol`		
Location(s)	transferOwnership		
Confirmed Fix At	6864054		

The Vertex Vault currently implements a one-step management transfer logic, where management can be transferred to another account in a single transaction. However, best practices recommend using a two-step management transfer logic for increased security and error recovery.

Impact With the current one-step management transfer logic, a mistake when transferring management to another account could be unrecoverable.

Recommendation Implement a two-step management transfer procedure for transferring management of the Vault. This should involve an initial step to initiate the transfer request and a second step to confirm and finalize the transfer. This approach adds an extra layer of security and allows for easier error recovery in case of mistakes during the transfer process.

Developer Response We have added an upgrade role to the vault to mitigate the unsafe manager role transfer.

4.1.7 V-VTX-VUL-007: Missing comments about implementation upgrade

Severity	Info	**Commit**	6864054
Type	Maintainability	**Status**	Fixed
File(s)	`RangeProtocolVertexVault.sol`		
Location(s)	See description		
Confirmed Fix At	6180e9b		

The `RangeProtocolVertexVault` contract is missing comments about how to properly upgrade the contract in case of a change on the implementation logic of the vault. This is critical as there is a bug that arises if the storage layout is not consistent between the old and new implementations.

Impact Without proper documentation on how to upgrade the implementation contract, there is a risk of introducing bugs related to inconsistent storage layouts. This could lead to unexpected behavior or vulnerabilities in the upgraded contract.

Recommendation It is recommended to add comments to the `RangeProtocolVertexVaultStorage` contract explaining how to update storage variables during an upgrade. This should include instructions on how to ensure the storage layout remains consistent between the old and new implementations, such as only appending state variables at the end of `RangeProtocolVertexVaultStorage` contract and to not modify the inheritance tree of the `RangeProtocolVertexVault` contract.

Developer Response added the comments.