Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Range Protocol

Vault Manager Contracts

Veridise Inc.
September 27, 2023

► **Prepared For:**

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

► **Prepared By:**

Alberto Gonzalez
Benjamin Sepanski

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

► **Version History:**

Sep. 19, 2023	Initial Draft

Contents

1 Executive Summary

From Sep. 4, 2023 to Sep. 13, 2023, Range Protocol engaged Veridise to review the security of their Vault Manager Contracts. These vaults are designed to store user funds. A manager is responsible for using user-supplied liquidity in Uniswap pools to produce profit in return for fees. The Veridise auditors reviewed the vault implementation and its associated factory, as well as slight modifications to Uniswap periphery contracts and renamings in an OpenZeppelin access-control contract.

Veridise conducted the assessment over 2 person-weeks, with 2 engineers reviewing code over 1 week on commit `0x0bc6281e`. 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 Vault Manager Contracts developers provided the source code of the Vault Manager Contracts 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 READMEs and documentation 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. Several files in the source code also indicate that the developers use linting such as prettier.

The Veridise auditors felt that the code was well organized and followed Solidity best practices. The vault had a very limited interface, which also reduces the attack surface. Note that this is under the assumption that the managers are a fully trusted entity, which was indicated to the auditors by the Range Protocol team.

Summary of issues detected. The audit uncovered 9 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. The Veridise auditors also identified 2 medium-severity issues, including retroactive application of fees (V-RNG-VUL-001) and lack of slippage protection (V-RNG-VUL-002). The audit also uncovered a number of minor issues, including an out-of-bounds array access (V-RNG-VUL-003) and opportunities for small theft under mismanagement (V-RNG-VUL-004), as well as several maintainability issues. The Vault Manager Contracts developers have resolved or acknowledged all 9 issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Vault Manager Contracts.

Primary amongst these is to implement slippage protection as described in V-RNG-VUL-002.

The auditors also recommend adding extensive documentation for managers. The Uniswap [*]core contracts are designed only to implement core functionality. Many issues (such as optimality and slippage protection) are implemented in the [†]periphery contracts. Managers may be prone to a host of mathematical or Solidity-specific errors handled in the periphery, and should be strongly recommended to rely on the heavily-used Uniswap contracts.

The auditors also recommend that this guide contain key management standards, as the manager is a trusted point in the protocol. The guide should also include the recommendation from issue V-RNG-VUL-004.

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.

[*] `https://github.com/Uniswap/v3-core`
[†] `https://github.com/Uniswap/v3-periphery/`

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Vault Manager Contracts	`0x0bc6281e` - `0x0bc6281`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 4 - Sep. 13, 2023	Manual & Tools	2	2 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	2	2
Low-Severity Issues	2	2
Warning-Severity Issues	3	3
Informational-Severity Issues	2	2
TOTAL	9	9

Table 2.4: Category Breakdown.

Name	Number
Logic Error	3
Data Validation	2
Maintainability	2
Transaction Ordering	1
Authorization	1

3 Audit Goals and Scope

3.1 Audit Goals

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

- Is it possible to reenter the vault?
- Can reentry from the Uniswap pool lead to theft from the vault?
- Can manipulation of the Uniswap price between interactions with the vault enable theft?
- 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?

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 `contracts/` folder of the source code provided by the Vault Manager Contracts developers, which contains the smart contract implementation of the Vault Manager Contracts. Namely, the `RangeProtocolFactory.sol`, `RangeProtocolVault.sol`, and `RangeProtocolStorage.sol` files, along with the `access/`, `errors/`, and `uniswap/` directories.

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

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

Vulnerability Report 4

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-RNG-VUL-001	Retroactive fees	Medium	Fixed
V-RNG-VUL-002	No slippage protection in mint(), swap(), or bu. . .	Medium	Fixed
V-RNG-VUL-003	Out-of-bounds array access	Low	Fixed
V-RNG-VUL-004	Poor management of fees may lead to small theft	Low	Acknowledged
V-RNG-VUL-005	Fee caps not checked at initialization	Warning	Fixed
V-RNG-VUL-006	Manager address could be zero	Warning	Fixed
V-RNG-VUL-007	Existing issues from prior reports	Warning	Acknowledged
V-RNG-VUL-008	Unused error	Info	Fixed
V-RNG-VUL-009	Performance fee cap documented incorrectly	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-RNG-VUL-001: Retroactive fees

Severity	Medium	**Commit**	0bc6281
Type	Logic Error	**Status**	Fixed
File(s)	`RangeProtocolVault.sol`		
Location(s)	updateFees()		
Confirmed Fix At	3f4e4f7		

The `updateFees()` function allows the manager to change the fees.

```
function updateFees(
    uint16 newManagingFee,
    uint16 newPerformanceFee
) external override onlyManager {
    if (newManagingFee > MAX_MANAGING_FEE_BPS) revert VaultErrors.InvalidManagingFee
    ();
    if (newPerformanceFee > MAX_PERFORMANCE_FEE_BPS) revert VaultErrors.
    InvalidPerformanceFee();

    managingFee = newManagingFee;
    performanceFee = newPerformanceFee;
    emit FeesUpdated(newManagingFee, newPerformanceFee);
}
```

Snippet 4.1: Implementation of `updateFees()`.

However, performance fees are not collected until fees are pulled, tokens are burned, or liquidity is removed. Thus, the new performance fee will be applied retroactively on fees garnered from liquidity in the Uniswap pool.

Impact Users may be forced to pay larger fees than indicated in the contract.

Recommendation Collect uniswap fees from the pool and apply the old performance fee before updating to the new fee.

Developer Response We acknowledged the issue and applied the recommended fix.

4.1.2 V-RNG-VUL-002: No slippage protection in mint(), swap(), or burn()

Severity	Medium	**Commit**	0bc6281
Type	Transaction Ordering	**Status**	Fixed
File(s)	`RangeProtocolVault.sol`		
Location(s)	mint(), burn()		
Confirmed Fix At	4120118		

Callers of `mint()` submit a `mintAmount`, and then the `RangeProtocolVault` computes how much of each token is owed. For example, the `totalSupply > 0` case is shown in the below code snippet.

```
(uint256 amount0Current, uint256 amount1Current) = getUnderlyingBalances();
amount0 = FullMath.mulDivRoundingUp(amount0Current, mintAmount, totalSupply);
amount1 = FullMath.mulDivRoundingUp(amount1Current, mintAmount, totalSupply);
```

Snippet 4.2: Computation of `(amount0, amount1)` when `totalSupply > 0` in `mint()`.

Once this is done, the specified amounts of each token is transferred to the pool.

```
if (amount0 > 0) {
    userVaults[msg.sender].token0 += amount0;
    token0.safeTransferFrom(msg.sender, address(this), amount0);
}
if (amount1 > 0) {
    userVaults[msg.sender].token1 += amount1;
    token1.safeTransferFrom(msg.sender, address(this), amount1);
}
```

Snippet 4.3: Transfer of funds inside `mint()`.

The amounts are never checked, and may be larger than the user expects. In order to protect against this, a user must have approved only a small amount.

Similarly, `burn()`, `swap()`, `addLiquidity()`, and `removeLiquidity()` have no slippage protection.

Impact Users have the largest potential impact with `mint()` and `burn()`. If a large `swap()` occurs on the Uniswap pool right before the `mint()` (resp. `burn()`), the value expended may become much larger (resp. smaller) than expected. For `mint()`, a user who naively approves `uint.max` for the pool may then pay more than they wish. For `burn()`, regardless of user action they may receive less than expected.

Swapping and adding/removing liquidity primarily affects the manager, who must be aware of the possibility that their action could be frontrun.

Recommendation We recommend adding a parameter to provide slippage protection.

If not possible, we suggest noting in the documentation that a user must approve only the amount they wish to spend for `mint()`, but still adding slippage protection for `burn()`. The developers should also document that the managers are expected to check for slippage.

Developer Response Slippage protection has been added.

Veridise Response We would recommend adding upper bounds for slippage protection when sending funds, i.e. in `mint()` and `addLiquidity()`.

Updated Developer Response We have updated the slippage protections.

4.1.3 V-RNG-VUL-003: Out-of-bounds array access

Severity	Low	**Commit**	0bc6281
Type	Logic Error	**Status**	Fixed
File(s)	`RangeProtocolFactory.sol`		
Location(s)	getVaultAddresses()		
Confirmed Fix At	3f4e4f7		

The function `getVaultAddresses()` in `RangeProtocolFactory.sol` allows callers to receive any contiguous subset of the `_vaultsList` from a `startIdx` through an `endIdx` by copying the values into a local `vaultList` array, and returning that to the user.

```
function getVaultAddresses(
        uint256 startIdx,
        uint256 endIdx
    ) external view returns (address[] memory vaultList) {
        vaultList = new address[](endIdx - startIdx + 1);
        for (uint256 i = startIdx; i <= endIdx; i++) {
            vaultList[i] = _vaultsList[i];
        }
    }
```

Snippet 4.4: Implementation of `getVaultAddresses()`

However, the `vaultList` is indexed at `i`, which ranges from `startIdx` through `endIdx`, rather than at `i-startIdx`, which ranges from `0` through `endIdx-startIdx`.

Impact Any call to `getVaultAddresses()` with `startIdx > 0` will cause an out-of-bounds array access in the last few iterations of the loop.

Recommendation Replace `vaultList[i] = _vaultsList[i];` with `vaultList[i-startIdx] = _vaultsList[i];`.

Developer Response We acknowledged the finding and the recommended fix has been applied.

4.1.4 V-RNG-VUL-004: Poor management of fees may lead to small theft

Severity	Low	**Commit**	0bc6281
Type	Logic Error	**Status**	Acknowledged
File(s)	RangeProtocolVault.sol		
Location(s)	burn(), _getUnderlyingBalances(), getBalanceInCollateralToken()		
Confirmed Fix At	N/A		

When calling `burn()` or `_computeUnderlyingBalances()`, the manager balance is subtracted from the accrued fees if the accrued fees are large enough.

```
(uint256 burn0, uint256 burn1, uint256 fee0, uint256 fee1) = _withdraw(
    liquidityBurned);

_applyPerformanceFee(fee0, fee1);
(fee0, fee1) = _netPerformanceFees(fee0, fee1);
emit FeesEarned(fee0, fee1);

uint256 passiveBalance0 = token0.balanceOf(address(this)) - burn0;
uint256 passiveBalance1 = token1.balanceOf(address(this)) - burn1;
if (passiveBalance0 > managerBalance0) passiveBalance0 -= managerBalance0;
if (passiveBalance1 > managerBalance1) passiveBalance1 -= managerBalance1;

amount0 = burn0 + FullMath.mulDiv(passiveBalance0, burnAmount, totalSupply);
amount1 = burn1 + FullMath.mulDiv(passiveBalance1, burnAmount, totalSupply);
```

Snippet 4.5: `inThePosition` case of `burn()`.

This means that, if the manager balance is very near, but below, the passive balance, a user who burns from the vault will receive more than expected.

Impact If managers keep enough liquid values to just barely cover their fees, they may be vulnerable to small thefts from the vault.

For example, the below scenario shows a situation where

1. An attacker mints in the vault.
2. The manager pulls fees from the pool then rebalances liquidity leaving just enough for fees.
3. The attacker burns their tokens, profiting from the exchange.

```
Current fees from pool: (0.200229,0.200004)
Attacker quickly mints before fees are pulled from the pool
Cost: (1.994866,1.994872), Tokens Minted: 2000000000000000000
Fees are pulled from pool
Vault balance:   (0.971597,1.136293)
Manager balance: (0.005134,0.005128)
Manager adds liquidity to the pool, leaving just enough for fees
Vault balance:   (0.005134,0.222477)
Manager balance: (0.005134,0.005128)
Attacker burns all tokens!
Attacker reward: (1.997433,1.994872)
Attacker profit: (0.002567,-0.000000)
```

Note that the amounts stolen here are typically small.

Recommendation We recommend either reverting when the passive balance cannot cover the debt to the manager, or at least emitting an event.

If choosing not to revert, the range protocol should be sure to inform managers of this possibility, and ensure the managers withdraw fees frequently.

Developer Response The off-chain strategy of managers will take the decision of either to keep the manager fee in the vault or deploy it on the pool when adding liquidity along with the assets.

4.1.5 V-RNG-VUL-005: Fee caps not checked at initialization

Severity	Warning	**Commit**	0bc6281
Type	Data Validation	**Status**	Fixed
File(s)	`RangeProtocolVault.sol`		
Location(s)	constructor()		
Confirmed Fix At	3f4e4f7		

In the `constructor()` of `RangeProtocolVault`, the `performanceFee` and `managingFee` are initialized. However, there is no comparison to `MAX_PERFORMANCE_FEE_BPS` or `MAX_MANAGING_FEE_BPS` to ensure the initialized values are in the proper range.

```
performanceFee = 250;
managingFee = 0;
// Managing fee is 0% at the time vault initialization.
emit FeesUpdated(0, performanceFee);
```

Snippet 4.6: Initialization of fees in the constructor.

Impact If initialization values are changed, a developer may unintentionally change the fees to be larger than their respective caps.

Recommendation Assert `performanceFee < MAX_PERFORMANCE_FEE_BPS` and `managingFee < MAX_MANAGING_FEE_BPS` in the constructor.

Developer Response We acknowledged the issue and applied the recommended fix.

4.1.6 V-RNG-VUL-006: Manager address could be zero

Severity	Warning	**Commit**	0bc6281
Type	Data Validation	**Status**	Fixed
File(s)	`RangeProtocolVault.sol`		
Location(s)	initialize()		
Confirmed Fix At	3f4e4f7		

The `manager` for the vault is provided through the `data` parameter of `initialize()`. Then, ownership is transferred to the `manager`.

```
function initialize(
    address _pool,
    int24 _tickSpacing,
    bytes memory data
) external override initializer {
    (address manager, string memory _name, string memory _symbol) = abi.decode(
        data,
        (address, string, string)
    );

    __UUPSUpgradeable_init();
    __ReentrancyGuard_init();
    __Ownable_init();
    __ERC20_init(_name, _symbol);
    __Pausable_init();

    _transferOwnership(manager);
```

Snippet 4.7: Implementation of `initialize()`.

The caller of initialize is `_createVault()` inside `RangeProtocolFactory`:

```
function _createVault(
    address tokenA,
    address tokenB,
    uint24 fee,
    address pool,
    address implementation,
    bytes memory data
) internal returns (address vault) {
    if (data.length == 0) revert FactoryErrors.NoVaultInitDataProvided();
    // .... irrelevant code elided
    vault = address(
        new ERC1967Proxy(
            implementation,
            abi.encodeWithSelector(INIT_SELECTOR, pool, tickSpacing, data)
        )
    );
    _vaultsList.push(vault);
}
```

Snippet 4.8: Implementation of `_createVault()`.

So, the only validation performed on `data` is length-based.

Impact The `manager` address may be zero without causing any errors during initialization.

Recommendation Revert during initialization if the `manager` address (or other user-supplied addresses) are `0x0`.

Developer Response We acknowledged the finding and applied the recommended fix.

4.1.7 V-RNG-VUL-007: Existing issues from prior reports

Severity	Warning	**Commit**	0bc6281
Type	Authorization	**Status**	Acknowledged
File(s)	N/A		
Location(s)	N/A		
Confirmed Fix At	N/A		

Here we highlight several issues identified by previous audit reports which remain open, and add our encouragement to that of prior auditors for the developers to address these issues.

1. **HAL-04: Users can steal any manually added liquidity.** In both repositories, if a manager adds liquidity manually, then vault tokens become more valuable and attackers may frontrun to buy the tokens while cheap, then burn them once they've gained value.
2. **HAL-05: Fee payment bypass is possible for small amounts.** Since fees are computed with only 4 decimals, small burn amounts may avoid fees.
3. **HAL-07: Malicious manager can steal a share of vault liquidity.** In both repositories, the managers are trusted entities. For instance, a manager may perform a large number of swaps with the users own funds simply to generate fees.

Impact

1. Manually added liquidity may go to waste.
2. Small fees may go unpaid.
3. Untrusted managers, or managers who are hacked, may use bad swaps to steal user funds.

Recommendation

1. Be sure to document this possibility and make it clear that managers should not add liquidity manually.
2. Notify managers of this possibility, or set a minimum fee.
3. Make clear to users that managers must be fully trusted, and add extra requirements on managers to ensure their keys are stored safely.

Developer Response The managers will be KYCed and we have an onboarding document for them. We will add the aforementioned points in the onboarding document.

4.1.8 V-RNG-VUL-008: Unused error

Severity	Info	**Commit**	0bc6281
Type	Maintainability	**Status**	Fixed
File(s)	`errors/VaultErrors.sol`		
Location(s)	library VaultErrors		
Confirmed Fix At	3f4e4f7		

The `MintFailed()` error defined in the `VaultErrors` library is unused.

Impact Future developers may be confused about which error type to use.

Recommendation Remove the unused error type.

Developer Response Acknowledge the finding and the unused error has been removed.

4.1.9 V-RNG-VUL-009: Performance fee cap documented incorrectly

Severity	Info	**Commit**	0bc6281
Type	Maintainability	**Status**	Fixed
File(s)	`RangeProtocolVault.sol`		
Location(s)	RangeProtocolVault		
Confirmed Fix At	3f4e4f7		

The README in the contracts repository indicates two different caps on fees:

> - Vault manager can update the managing and performance fee managing fee is capped at 1% and performance fee is capped at 1%.
>
> There are two types of fees i.e. performance fee and managing fee. Performance fee will be capped at 10% (1000 BPS) and at the time of vault initialisation, it will be set to 250 BPS (2.5%). The managing fee at the time of vault initialisation will be set to 0%, but it can be set up to 1% (100 BPS). Both of these fees are credited to state variables of `managerBalance0` and `managerBalance1`.

In `RangeProtocolVault.sol`, the caps are set as constants:

```
/// Performance fee cannot be set more than 10% of the fee earned from uniswap v3
    pool.
uint16 public constant MAX_PERFORMANCE_FEE_BPS = 1000;
/// Managing fee cannot be set more than 1% of the total fee earned.
uint16 public constant MAX_MANAGING_FEE_BPS = 100;
```

Snippet 4.9: Fee caps defined in `RangeProtocolVault.sol`.

Impact Users of the protocol who read the documentation may misunderstand what the fee caps are.

Recommendation Fix the documentation to list the performance fee cap at 10%.

Developer Response We acknowledge the finding and the documentation has been corrected.

Glossary

AMM Automated Market Maker. 21

OpenZeppelin A security company which provides many standard implementations of common contract specifications. See `https://www.openzeppelin.com`. 1

prettier A code formatting tool, see `https://prettier.io/docs/en/integrating-with-linters.html` to learn more. 1

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 21

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 2

Uniswap One of the most famous deployed AMMs. See `https://uniswap.org` to learn more. 1