HALBORN

Range Protocol - Active Liquidity Management

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 15th, 2023 - June 5th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/15/2023	Grzegorz Trawinski
0.2	Document Updates	05/17/2023	Grzegorz Trawinski
0.3	Document Updates	06/02/2023	Isabel Burruezo
0.4	Draft Version	06/02/2023	Manuel Diaz
0.5	Draft Review	06/05/2023	Grzegorz Trawinski
0.6	Draft Review	06/05/2023	Ataberk Yavuzer
0.7	Draft Review	06/06/2023	Piotr Cielas
0.8	Draft Review	06/06/2023	Gabi Urrutia
1.0	Remediation Plan	06/21/2023	Isabel Burruezo
1.1	Remediation Plan	06/22/2023	Grzegorz Trawinski
1.2	Remediation Plan Review	06/22/2023	Ataberk Yavuzer
1.3	Remediation Plan Review	06/22/2023	Piotr Cielas
1.4	Remediation Plan Review	06/23/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com
Manuel Garcia Diaz	Halborn	Manuel.Diaz@halborn.com
Isabel Burruezo	Halborn	Isabel.Burruezo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Range Protocol provides permissionless infrastructure for smart money management, bringing maximized yields and optimal capital efficiency to users of different risk profiles.

Range Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on May 15th, 2023 and ending on June 5th, 2023. The security assessment was scoped to the smart contracts provided in the contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided 3 weeks for the engagement and assigned 3 full-time security engineers to audit the security of the smart contracts in scope. The security engineers are blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audit is to:

- Identify potential security issues within the smart contracts
- Verify whether Factory and Vaults work as expected

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks. The majority of findings were assigned medium or low-risk rate. The team at Halborn received the updated code base on 20th of June with the applied remediation. The Range Protocol team addressed and accepted the risk in findings. The risks will be managed outside the realm of smart contracts by Range Protocol and vaults' managers.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Active Liquidity Management - uniswap - master branch

- Repository: contracts
- Commit ID: 2d1a6334139ed9d6c60ff44e16c5a4198ebab737
- Branch: master
- Smart contracts in scope:
  1. /contracts/RangeProtocolVault.sol
  2. /contracts/RangeProtocolVaultStorage.sol
  3. /contracts/RangeProtocolFactory.sol
  4. /contracts/interfaces/IRangeProtocolFactory.sol
  5. /contracts/interfaces/IRangeProtocolVault.sol

Apart from the master branch, the changes introduced in the below pull requests were also included in the scope:

- https://github.com/Range-Protocol/contracts/pull/3/files
- https://github.com/Range-Protocol/contracts/pull/4/files

The details of these two source branches are provided below.

2. Active Liquidity Management - pancake

- Repository: contracts
- Commit ID: 15df0530e83ed3482e7062356f0fd2bc8fcd7e8b
- Branch: implement-pancake-swap-compatability
- Smart contracts in scope:
  1. /contracts/RangeProtocolVault.sol
  2. /contracts/RangeProtocolVaultStorage.sol

3. /contracts/RangeProtocolFactory.sol
4. /contracts/interfaces/IRangeProtocolFactory.sol
5. /contracts/interfaces/IRangeProtocolVault.sol

3. Active Liquidity Management - algebra

- Repository: contracts
- Commit ID: 0584307c08514e68bcaeb31d0601564140fcb70b
- Branch: implement-algebra-compatability
- Smart contracts in scope:
  1. /contracts/RangeProtocolVault.sol
  2. /contracts/RangeProtocolVaultStorage.sol
  3. /contracts/RangeProtocolFactory.sol
  4. /contracts/interfaces/IRangeProtocolFactory.sol
  5. /contracts/interfaces/IRangeProtocolVault.sol

Retest

During the retest phase, the Range Protocol team provided the following commits with fixes:

- 62abb0f21eed6b2608b9c2883780bb364f24f8bd, branch:fix_user_vault
- a883d28c7a8afe48201c54f200c58379995b4e46, branch: fix/add-liquidity
- f0777366d8204d81cb9ac2bf4b4b21b8d0be6728, branch:halborn-fix-hal-09

Out-of-Scope:
- /contracts/access
- /contracts/errors
- /contracts/mock
- /contracts/uniswap
- /contracts/algebra
- /contracts/pancake
- third-party libraries and dependencies
- economic attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	6	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING STORAGE GAPS IN UPGRADEABLE CONTRACT	Medium (5.9)	RISK ACCEPTED
RANGEPROTOCOLFACTORY LACKS OWNERSHIP-TRANSFER PATTERN	Low (3.0)	RISK ACCEPTED
OWNERSHIP CAN BE RENOUNCED IN RANGEPROTOCOLFACTORY	Low (2.7)	RISK ACCEPTED
USERS CAN STEAL ANY MANUALLY ADDED LIQUIDITY	Low (2.5)	RISK ACCEPTED
FEE PAYMENT BYPASS IS POSSIBLE FOR SMALL AMOUNTS	Low (2.5)	RISK ACCEPTED
USERVAULTS AND USERS ARE NOT UPDATED ON TOKENS TRANSFER	Low (2.5)	SOLVED - 06/22/2023
MALICIOUS MANAGER CAN STEAL A SHARE OF VAULT LIQUIDITY	Low (2.0)	RISK ACCEPTED
UPGRADETOANDCALL IS NOT SUPPORTED BY RANGEPROTOCOLFACTORY	Informational (1.2)	ACKNOWLEDGED
EDGE CASE NOT HANDLED WHEN LIQUIDITY IS ADDED WITHOUT PREVIOUSLY COLLECTING MANAGER FEES	Informational (1.4)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISSING STORAGE GAPS IN UPGRADEABLE CONTRACT - MEDIUM (5.9)

Description:

For upgradeable contracts, there must be storage gaps implemented to allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments.
As base contracts are stored first in the storage layout of the contract that inherits from them, upgrading a base contract adding new state variables or data structures could lead to data loss or corruption. So, that if the upgraded contract adds new variables or data structures without leaving enough unused storage slots, it could overwrite existing data, potentially causing the contract to malfunction or behave unexpectedly.
By including storage gaps, a contract can be designed to allow for future upgrades without risking the integrity of the contract's data.

Code Location:

Listing 1: RangeProtocolVaultStorage.sol (Line 48)

```
40 contract RangeProtocolVault is
41     Initializable,
42     UUPSUpgradeable,
43     ReentrancyGuardUpgradeable,
44     OwnableUpgradeable,
45     ERC20Upgradeable,
46     PausableUpgradeable,
47     IRangeProtocolVault,
48     RangeProtocolVaultStorage
49 {
```

Listing 2: RangeProtocolVaultStorage.sol

```
10 abstract contract RangeProtocolVaultStorage {
11     int24 public lowerTick;
12     int24 public upperTick;
13     bool public inThePosition;
```

```
    bool public mintStarted;
    int24 public tickSpacing;
    IUniswapV3Pool public pool;
    IERC20Upgradeable public token0;
    IERC20Upgradeable public token1;
    address public factory;
    uint16 public managingFee;
    uint16 public performanceFee;
    uint256 public managerBalance0;
    uint256 public managerBalance1;
    struct UserVault {
        bool exists;
        uint256 token0;
        uint256 token1;
    }
    mapping(address => UserVault) public userVaults;
    address[] public users;
    // NOTE: Only add more state variable below it and do not
    ↳ change the order of above state variables.
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:H/D:N/Y:N/R:P/S:C (5.9)

Recommendation:

Consider including the proper storage gaps in RangeProtocolVaultStorage according to the base contracts that are meant to be upgradeable.

Remediation Plan:

RISK ACCEPTED: The Range Protocol team accepts this risk, since the RangeProtocolVault contract will not be inherited by another contract or act as a derived contract to another base contract in the future. Thus, any addition of new state variables in the contract could be done without the presence of storage gap.

4.2 (HAL-02) RANGEPROTOCOLFACTORY LACKS OWNERSHIP-TRANSFER PATTERN - LOW (3.0)

Description:

The RangeProtocolFactory contract implements the Ownable pattern. However, the assessment revealed that the solution does not support the two-step ownership-transfer pattern. The ownership transfer might be accidentally set to an inactive EOA account. In the case of account hijacking, multiple functionalities get under permanent control of the attacker, including createVault() and upgradeVault() functions.

Code Location:

Listing 3: RangeProtocolFactory.sol

```
/**
 * @dev Mars@RangeProtocol
 * @notice RangeProtocolFactory deploys and upgrades proxies for
↳ Range Protocol vault contracts.
 * Owner can deploy and upgrade vault contracts.
 */
contract RangeProtocolFactory is IRangeProtocolFactory, Ownable {
    bytes4 public constant INIT_SELECTOR =
        bytes4(keccak256(bytes("initialize(address,int24,bytes)"))
↳ );
(...)
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:H/A:H/D:M/Y:M/R:N/S:C (3.0)

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account.

Remediation Plan:

RISK ACCEPTED: The Range Protocol team plans to move the factory's ownership to the Timelock contract. The planned implementation of such a contract assumes at least 24 hours of execution delay controlled by a multi-signature wallet of at least five signers, with a quorum of three required. Usage of Ownable is preferred.

4.3 (HAL-03) OWNERSHIP CAN BE RENOUNCED IN RANGEPROTOCOLFACTORY - LOW (2.7)

Description:

The RangeProtocolFactory contract implements the Ownable pattern. However, the assessment revealed that the solution supports the renounceOwnership() function. Renouncing ownership prevents calling any significant functionality from the factory, including createVault() and upgradeVault() functions.

Code Location:

Listing 4: Ownable.sol

```
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:H/A:H/D:L/Y:L/R:N/S:C (2.7)

Recommendation:

It is recommended that the owner cannot call the renounceOwnership() function without transferring the ownership to another address.

Remediation Plan:

RISK ACCEPTED: The Range Protocol accepted the risk of this finding. The Range Protocol team does not plan to change the ownership once the ownership is moved to the Timelock controlled by Range Owner Multisig. Any changes in ownership will be done through change in signers of the multisig.

4.4 (HAL-04) USERS CAN STEAL ANY MANUALLY ADDED LIQUIDITY - LOW (2.5)

Description:

The RangeProtocolVault contract allows adding liquidity to an Uniswap V3 position from the vault's balance. This liquidity usually comes from the user's minted balance.

The shares are calculated based on the position's liquidity and vault's balance.

However, this means that if the manager decides to manually add funds to the vault, a user would be able to back-run the transaction, mint vault shares and burn them immediately after, which would allow them to drain the liquidity the manager manually introduced.

Code Location:

Listing 5: RangeProtocolVault.sol

```
    if (totalSupply > 0) {
        //@audit-issue > 0 consumes more gas.
        (
            uint256 amount0Current,
            uint256 amount1Current
        ) = getUnderlyingBalances();
        amount0 = FullMath.mulDivRoundingUp(
            amount0Current,
            mintAmount,
            totalSupply
        );
        amount1 = FullMath.mulDivRoundingUp(
            amount1Current,
            mintAmount,
            totalSupply
        );
    } else if (_inThePosition) {
        // If total supply is zero then inThePosition must be set
```

```
to accept token0 and token1 based on currently set ticks.
        // This branch will be executed for the first mint and as well as each time total supply is to be changed from zero to non-zero.
        (amount0, amount1) = LiquidityAmounts.getAmountsForLiquidity(
            sqrtRatioX96,
            lowerTick.getSqrtRatioAtTick(),
            upperTick.getSqrtRatioAtTick(),
            SafeCastUpgradeable.toUint128(mintAmount)
        );
    } else
```

Proof of Concept:

Listing 6

```
function test_Manu_StealLiquidity() public {
    deal(address(tokenA), address(ALICE), 1 ether);
    deal(address(tokenB), address(ALICE), 1 ether);
    deal(address(tokenA), address(BOB), 500 ether);
    deal(address(tokenB), address(BOB), 500 ether);

    deal(address(tokenA), address(vault), 1_000 ether);
    deal(address(tokenB), address(vault), 1_000 ether);

    vault.updateTicks(-200, 200);
    vault.updateFees(0, 1000);

    vault.addLiquidity(
        -200,
        200,
        tokenA.balanceOf(address(vault)),
        tokenB.balanceOf(address(vault))
    );

    console2.log("Alice initial balance:");
    console2.log(tokenA.balanceOf(ALICE));
    console2.log(tokenB.balanceOf(ALICE));

    vm.startPrank(ALICE);
    {
        tokenA.approve(address(vault), type(uint256).max);
        tokenB.approve(address(vault), type(uint256).max);
        vault.mint(1 ether);

        console2.log("Alice balance after mint:");
        console2.log(tokenA.balanceOf(ALICE));
        console2.log(tokenB.balanceOf(ALICE));

        vault.burn(vault.balanceOf(ALICE));

        console2.log("Alice balance after burn:");
        console2.log(tokenA.balanceOf(ALICE));
        console2.log(tokenB.balanceOf(ALICE));
    }
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended that the manager does not add liquidity manually, or change the share calculation formula.

Remediation Plan:

RISK ACCEPTED: The Range Protocol accepted the risk of this finding. To avoid this attack vector, the Range Protocol team will bootstrap all the vaults with some liquidity before they are indexed on the frontend and visible to general users. The adding of liquidity is only supposed to be done through the mint function. The addLiquidity() function will be used only for rebalancing the already minted liquidity, so there will not be manual liquidity provision in the vault. In scenarios where someone moves significant assets to vault directly mistakenly, the Range DAO can later vote on a proposal to pause the vault and work with the vault manager to reimburse the user by a change in vault logic.

4.5 (HAL-05) FEE PAYMENT BYPASS IS POSSIBLE FOR SMALL AMOUNTS - LOW (2.5)

Description:

The RangeProtocolVault contract is a vault solution that collects both performance and management fees. The management fee is collected with the burn() operation. However, due to rounding, it is possible to bypass fee payment when a small amount is burnt. Within the equation, the divisor is set to 10_000, whereas MAX_MANAGING_FEE_BPS is set to 100. Assuming that managingFee is set to 100, to bypass management fee, the user must burn up to around 100 tokens. The loss is rather negligible, but it might be more significant for ERC20 tokens with fewer decimals, e.g. 6.

Code Location:

Listing 7: RangeProtocolVault.sol

```
/**
     * @notice _applyManagingFee applies the managing fee to the
 ↳ notional value of the redeeming user.
     * @param amount0 user's notional value in token0
     * @param amount1 user's notional value in token1
     */
    function _applyManagingFee(uint256 amount0, uint256 amount1)
 ↳ private {
        uint256 _managingFee = managingFee;
        managerBalance0 += (amount0 * _managingFee) / 10_000;
        managerBalance1 += (amount1 * _managingFee) / 10_000;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended either to update the `_applyManagingFee()` function to remove the rounding issue or introduce a minimum `burn` value.

Remediation Plan:

RISK ACCEPTED: The Range Protocol team accepted the risk of this finding.

4.6 (HAL-06) USERVAULTS AND USERS ARE NOT UPDATED ON TOKENS TRANSFER - LOW (2.5)

Description:

The RangeProtocolVault contract inherits from the ERC20Upgradeable contract. During the mint() and burn() function calls, the userVaults and users collections are updated. However, no similar action is done when transfer() or transferFrom() functions are called. Thus, whenever users transfer tokens between each other, the data stored in the userVaults and users collections is incorrect. This data is used off-chain by the vault's manager whenever there is a need to estimate the exposure to maintain while rebalancing the position.

Code Location:

Listing 8: RangeProtocolVault.sol (Lines 203,204,207,211,277-279,283-285)

```
163 /**
164      * @notice mint mints range vault shares, fractional shares of
 ↳  a Uniswap V3 position/strategy
165      * to compute the amount of tokens necessary to mint `
 ↳ mintAmount` see getMintAmounts
166      * @param mintAmount The number of shares to mint
167      * @return amount0 amount of token0 transferred from msg.
 ↳ sender to mint `mintAmount`
168      * @return amount1 amount of token1 transferred from msg.
 ↳ sender to mint `mintAmount`
169      */
170     function mint(
171         uint256 mintAmount
172     ) external override nonReentrant whenNotPaused returns (
 ↳ uint256 amount0, uint256 amount1) {
173         if (!mintStarted) revert VaultErrors.MintNotStarted();
174         if (mintAmount == 0) revert VaultErrors.InvalidMintAmount
 ↳ ();
```

```
        uint256 totalSupply = totalSupply();
        bool _inThePosition = inThePosition;
        (uint160 sqrtRatioX96, , , , , , ) = pool.slot0();

        if (totalSupply > 0) {
            (uint256 amount0Current, uint256 amount1Current) = getUnderlyingBalances();
            amount0 = FullMath.mulDivRoundingUp(amount0Current, mintAmount, totalSupply);
            amount1 = FullMath.mulDivRoundingUp(amount1Current, mintAmount, totalSupply);
        } else if (_inThePosition) {
            // If total supply is zero then inThePosition must be set to accept token0 and token1 based on currently set ticks.
            // This branch will be executed for the first mint and as well as each time total supply is to be changed from zero to non-zero.
            (amount0, amount1) = LiquidityAmounts.getAmountsForLiquidity(
                sqrtRatioX96,
                lowerTick.getSqrtRatioAtTick(),
                upperTick.getSqrtRatioAtTick(),
                SafeCastUpgradeable.toUint128(mintAmount)
            );
        } else {
            // If total supply is zero and the vault is not in the position then mint cannot be accepted based on the assumptions
            // that being out of the pool renders currently set ticks unusable and totalSupply being zero does not allow
            // calculating correct amounts of amount0 and amount1 to be accepted from the user.
            // This branch will be executed if all users remove their liquidity from the vault i.e. total supply is zero from non-zero and
            // the vault is out of the position i.e. no valid tick range to calculate the vault's mint shares.
            // Manager must call initialize function with valid tick ranges to enable the minting again.
            revert VaultErrors.MintNotAllowed();
        }

        if (!userVaults[msg.sender].exists) {
            userVaults[msg.sender].exists = true;
            users.push(msg.sender);
        }
        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);
        }

        _mint(msg.sender, mintAmount);
        if (_inThePosition) {
            uint128 liquidityMinted = LiquidityAmounts.getLiquidityForAmounts(
                sqrtRatioX96,
                lowerTick.getSqrtRatioAtTick(),
                upperTick.getSqrtRatioAtTick(),
                amount0,
                amount1
            );
            pool.mint(address(this), lowerTick, upperTick, liquidityMinted, "");
        }

        emit Minted(msg.sender, mintAmount, amount0, amount1);
    }

    /**
     * @notice burn burns range vault shares (shares of a Uniswap V3 position) and receive underlying
     * @param burnAmount The number of shares to burn
     * @return amount0 amount of token0 transferred to msg.sender for burning {burnAmount}
     * @return amount1 amount of token1 transferred to msg.sender for burning {burnAmount}
     */
    function burn(
        uint256 burnAmount
    ) external override nonReentrant whenNotPaused returns (uint256 amount0, uint256 amount1) {
        if (burnAmount == 0) revert VaultErrors.InvalidBurnAmount();
        uint256 totalSupply = totalSupply();
        uint256 balanceBefore = balanceOf(msg.sender);
        _burn(msg.sender, burnAmount);

        if (inThePosition) {
            (uint128 liquidity, , , , ) = pool.positions(
↳ getPositionID());
            uint256 liquidityBurned_ = FullMath.mulDiv(burnAmount,
↳  liquidity, totalSupply);
            uint128 liquidityBurned = SafeCastUpgradeable.
↳ toUint128(liquidityBurned_);
            (uint256 burn0, uint256 burn1, uint256 fee0, uint256
↳ fee1) = _withdraw(liquidityBurned);

            _applyPerformanceFee(fee0, fee1);
            (fee0, fee1) = _netPerformanceFees(fee0, fee1);
            emit FeesEarned(fee0, fee1);
            amount0 =
                burn0 +
                FullMath.mulDiv(
                    token0.balanceOf(address(this)) - burn0 -
↳ managerBalance0,
                    burnAmount,
                    totalSupply
                );

            amount1 =
                burn1 +
                FullMath.mulDiv(
                    token1.balanceOf(address(this)) - burn1 -
↳ managerBalance1,
                    burnAmount,
                    totalSupply
                );
        } else {
            (uint256 amount0Current, uint256 amount1Current) =
↳ getUnderlyingBalances();
            amount0 = FullMath.mulDiv(amount0Current, burnAmount,
↳ totalSupply);
            amount1 = FullMath.mulDiv(amount1Current, burnAmount,
↳ totalSupply);
        }

        _applyManagingFee(amount0, amount1);
        (uint256 amount0AfterFee, uint256 amount1AfterFee) =
↳ _netManagingFees(amount0, amount1);
        if (amount0 > 0) {
            userVaults[msg.sender].token0 =
                (userVaults[msg.sender].token0 * (balanceBefore -
↳ burnAmount)) /
                balanceBefore;
            token0.safeTransfer(msg.sender, amount0AfterFee);
        }
        if (amount1 > 0) {
            userVaults[msg.sender].token1 =
                (userVaults[msg.sender].token1 * (balanceBefore -
↳ burnAmount)) /
                balanceBefore;
            token1.safeTransfer(msg.sender, amount1AfterFee);
        }

        emit Burned(msg.sender, burnAmount, amount0AfterFee,
↳ amount1AfterFee);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to either prevent users from transferring vault's tokens or to update all relevant collection's states while transferring the tokens.

Remediation Plan:

SOLVED: The Range Protocol team solved this finding in commit 62abb0f2. The tokens transfer now updates all relevant collection states with the _beforeTokenTransfer() internal function.

4.7 (HAL-07) MALICIOUS MANAGER CAN STEAL A SHARE OF VAULT LIQUIDITY - LOW (2.0)

Description:

The RangeProtocolVault contract is a vault solution managed by a user with the manager role. The vault collects the users' liquidity, that is later used within dedicated the UniswapV3Pool. The assessment revealed that manager can steal users' liquidity by means of the removeLiquidity(), addLiquidity() and swap() functions.
Each swap() function call generates fees within the UniswapV3Pool contract. While calling the removeLiquidity() function, all tokens are transferred from the UniswapV3Pool into the vault. Also, the _applyPerformanceFee() function is called to update manager's fees. By means of addLiquidity(), the manager can transfer any amount of tokens from the vault to the UniswapV3Pool.
Ultimately, the malicious manager can remove all liquidity from the pool, then send part of it back to the pool, and use the remaining liquidity to perform multiple swap operations that generate fees.
Also, the RangeProtocolVault implements PausableUpgradeable, thus, prior to an attack, the malicious manager can pause the contract, preventing the legitimate users from calling the burn() function to withdraw deposits.

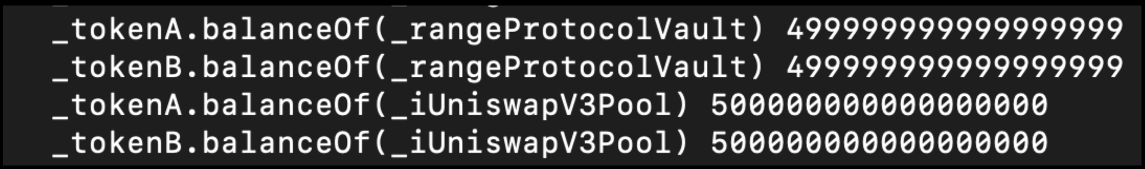
Code Location:

Listing 9: RangeProtocolVault.sol (Line 306)

```
/**
     * @notice removeLiquidity removes liquidity from uniswap pool
  ↳ and receives underlying tokens
     * in the vault contract.
     */
    function removeLiquidity() external override onlyManager {
        (uint128 liquidity, , , , ) = pool.positions(getPositionID());

        if (liquidity > 0) {
            int24 _lowerTick = lowerTick;
            int24 _upperTick = upperTick;
            (uint256 amount0, uint256 amount1, uint256 fee0, uint256 fee1) = _withdraw(liquidity);

            emit LiquidityRemoved(liquidity, _lowerTick, _upperTick, amount0, amount1);

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

        // TicksSet event is not emitted here since the emitting would create a new position on subgraph but
        // the following statement is to only disallow any liquidity provision through the vault unless done
        // by manager (taking into account any features added in future).
        lowerTick = upperTick;
        inThePosition = false;
        emit InThePositionStatusSet(false);
    }

    /**
     * @dev Mars@RangeProtocol
     * @notice swap swaps token0 for token1 (token0 in, token1 out), or token1 for token0 (token1 in token0 out).
     * Zero for one will cause the price: amount1 / amount0 lower, otherwise it will cause the price higher
     * @param zeroForOne The direction of the swap, true is swap token0 for token1, false is swap token1 to token0
     * @param swapAmount The exact input token amount of the swap
     * @param sqrtPriceLimitX96 threshold price ratio after the swap.
     * If zero for one, the price cannot be lower (swap make price lower) than this threshold value after the swap
     * If one for zero, the price cannot be greater (swap make price higher) than this threshold value after the swap
     * @return amount0 If positive represents exact input token0 amount after this swap, msg.sender paid amount,
     * or exact output token0 amount (negative), msg.sender received amount
     * @return amount1 If positive represents exact input token1 amount after this swap, msg.sender paid amount,
     * or exact output token1 amount (negative), msg.sender received amount
     */
    function swap(
        bool zeroForOne,
        int256 swapAmount,
        uint160 sqrtPriceLimitX96
    ) external override onlyManager returns (int256 amount0, int256 amount1) {
        (amount0, amount1) = pool.swap(
            address(this),
            zeroForOne,
            swapAmount,
            sqrtPriceLimitX96,
            ""
        );

        emit Swapped(zeroForOne, amount0, amount1);
    }

    /**
     * @dev Mars@RangeProtocol
     * @notice addLiquidity allows manager to add liquidity into uniswap pool into newer tick ranges.
     * @param newLowerTick new lower tick to deposit liquidity into
     * @param newUpperTick new upper tick to deposit liquidity into
     * @param amount0 max amount of amount0 to use
     * @param amount1 max amount of amount1 to use
     * @return remainingAmount0 remaining amount from amount0
     * @return remainingAmount1 remaining amount from amount1
     */
    function addLiquidity(
        int24 newLowerTick,
        int24 newUpperTick,
        uint256 amount0,
        uint256 amount1
    ) external override onlyManager returns (uint256 remainingAmount0, uint256 remainingAmount1) {
        _validateTicks(newLowerTick, newUpperTick);
        (uint160 sqrtRatioX96, , , , , , ) = pool.slot0();
        uint128 baseLiquidity = LiquidityAmounts.getLiquidityForAmounts(
            sqrtRatioX96,
            newLowerTick.getSqrtRatioAtTick(),
            newUpperTick.getSqrtRatioAtTick(),
            amount0,
            amount1
        );

        if (baseLiquidity > 0) {
            (uint256 amountDeposited0, uint256 amountDeposited1) = pool.mint(
                address(this),
                newLowerTick,
                newUpperTick,
                baseLiquidity,
                ""
            );
            // Should return remaining token number for swap
            remainingAmount0 = amount0 - amountDeposited0;
            remainingAmount1 = amount1 - amountDeposited1;
            if (lowerTick != newLowerTick || upperTick != newUpperTick) {
                lowerTick = newLowerTick;
                upperTick = newUpperTick;
                emit TicksSet(newLowerTick, newUpperTick);
            }

            emit LiquidityAdded(
                baseLiquidity,
                newLowerTick,
                newUpperTick,
                amountDeposited0,
                amountDeposited1
            );
        }
        // This check is added to not update inThePosition state in case manager decides to add liquidity in smaller chunks.
        if (!inThePosition) {
            inThePosition = true;
            emit InThePositionStatusSet(true);
        }
    }
```

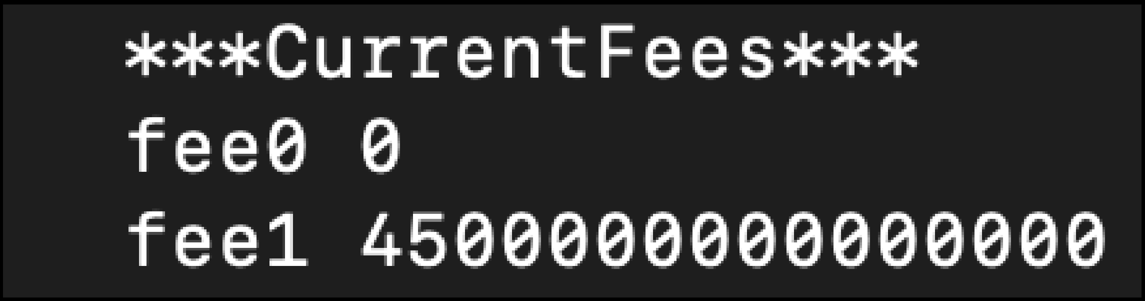
Proof of Concept:

1. As a depositor, mint() 1 ether of tokens.
2. As a manager, call the pause() function.
3. As a manager, set the performance fee to 10%.
4. As a manager, call the removeLiquidity() function. Observe that all tokens are transferred from the pool into the vault.
5. As a manager, call the addLiquidity() function for half of the available tokens.



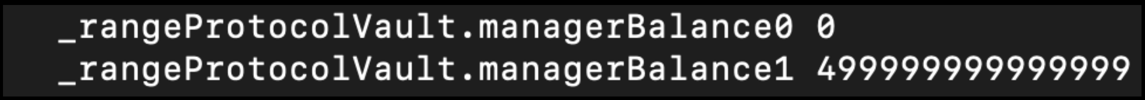
```
_tokenA.balanceOf(_rangeProtocolVault) 499999999999999999
_tokenB.balanceOf(_rangeProtocolVault) 499999999999999999
_tokenA.balanceOf(_iUniswapV3Pool) 500000000000000000
_tokenB.balanceOf(_iUniswapV3Pool) 500000000000000000
```

4. As a manager, call the swap() function for all tokenA available in the vault. Observe that pool's fees are accumulated.



```
***CurrentFees***
fee0 0
fee1 4500000000000000
```

5. As a manager, call the removeLiquidity() function. Observe that fees are transferred to the vault. Note that managerBalance variable has now accumulated value.



```
_rangeProtocolVault.managerBalance0 0
_rangeProtocolVault.managerBalance1 449999999999999
```

6. As a manager, call the collectManager() to collect fees.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to prevent managers from executing malicious action such as stealing the users' liquidity.

Remediation Plan:

RISK ACCEPTED: The Range Protocol accepted the risk of this finding. The team plans to cooperate only with sophisticated trading partners. Each partner will undergo extensive KYC (Know Your Customer) and AML (Anti Money Laundering) reviews done in prior to onboarding. Legal actions are planned against any malicious managers.

4.8 (HAL-08) UPGRADETOANDCALL IS NOT SUPPORTED BY RANGEPROTOCOLFACTORY - INFORMATIONAL (1.2)

Description:

The RangeProtocolFactory contract supports vault upgrade by means of the upgradeTo() function. However, it does not support the upgradeToAndCall() function. Vault can be upgraded only by means of the factory, so calling upgradeToAndCall() manually is not an option. Therefore, upgrading vault with an immediate call to the initialize function in a single transaction is not possible. In the event of an initializing next vault's version need, such call must be done separately. Depending on the implementation, such an approach might be vulnerable to various issues, including front-running, human-error or lack of initialization.

Code Location:

Listing 10: RangeProtocolFactory.sol

```
    function _upgradeVault(address _vault, address _impl) internal
 ↳  {
        (bool success, ) = _vault.call(abi.encodeWithSelector(
 ↳ UPGRADE_SELECTOR, _impl));

        if (!success) revert FactoryErrors.VaultUpgradeFailed();
        emit VaultImplUpgraded(_vault, _impl);
    }
```

Listing 11: RangeProtocolVault.sol

```
    function _authorizeUpgrade(address) internal override {
        if (msg.sender != factory) revert VaultErrors.
 ↳ OnlyFactoryAllowed();
    }
```

BVSS:

AO:S/AC:L/AX:L/C:L/I:L/A:L/D:L/Y:L/R:N/S:C (1.2)

Recommendation:

It is recommended to add support for upgradeToAndCall() in the RangeProtocolFactory.

Remediation Plan:

ACKNOWLEDGED: The Range Protocol team acknowledged this finding. The owner of the factory should not have any initialization control for state variables. Owner should have only implementation upgrade possibility. Any state variables change after upgrade must be done by the vault's manager.

4.9 (HAL-09) EDGE CASE NOT HANDLED WHEN LIQUIDITY IS ADDED WITHOUT PREVIOUSLY COLLECTING MANAGER FEES - INFORMATIONAL (1.4)

Description:

The burn() function allows anyone to burn shares from the vault and receive underlyings.
In addition, **performanceFees**, **ManagerBalance0** and **ManagerBalance1** fees are calculated and updated. The latter can be collected when the manager calls the collectManager() function.
On the other hand, the addLiquidity() function allows the manager to add the liquidity of the RangeProtocol contract to the uniswap pool at more recent tick ranges.

It has been detected that in case fees are not collected after one or more burns, and then liquidity is added, the protocol malfunctions. This is due to the fact that adding liquidity is done with the manager's fees that have not been collected and are stored in the contract, so it remains at 0.
- If collectManager() is called, there are no funds to transfer.
- If mint is called, it throws an underflow in the calculation of **amountCurrent0** and **amountCurrent1** in the call to _getUnderlyingBalance since the **managerBalance0** and **managerBalance1** are high, and the contract balance is 0.
- If burn() is called, it throws an underflow in the calculation of amount0 and amount1 since the value of **managerBalance0** and **managerBalance1** are the highest values in the subtraction.

This continues to happen until the manager calls removeLiquidity() and users call mint or burn again in case they minted before adding liquidity. And only after this, it is possible to get corresponding fees from the manager.

Code Location:

Listing 12: RangeProtocolVault.sol (Lines 448,451)

```
441 function collectManager() external override onlyManager {
442         uint256 amount0 = managerBalance0;
443         uint256 amount1 = managerBalance1;
444         managerBalance0 = 0;
445         managerBalance1 = 0;
446 
447         if (amount0 > 0) {
448             token0.safeTransfer(manager(), amount0);
449         }
450         if (amount1 > 0) {
451             token1.safeTransfer(manager(), amount1);
452         }
453     }
```

Listing 13: RangeProtocolVault.sol (Lines 260-263,269-272)

```
242 function burn(
243         uint256 burnAmount
244     ) external override nonReentrant whenNotPaused returns (
 ↳ uint256 amount0, uint256 amount1) {
245         if (burnAmount == 0) revert VaultErrors.InvalidBurnAmount
 ↳ ();
246         uint256 totalSupply = totalSupply();
247         uint256 balanceBefore = balanceOf(msg.sender);
248         _burn(msg.sender, burnAmount);
249 
250         if (inThePosition) {
251             (uint128 liquidity, , , , ) = pool.positions(
 ↳ getPositionID());
252             uint256 liquidityBurned_ = FullMath.mulDiv(burnAmount,
 ↳  liquidity, totalSupply);
253             uint128 liquidityBurned = SafeCastUpgradeable.
 ↳ toUint128(liquidityBurned_);
254             (uint256 burn0, uint256 burn1, uint256 fee0, uint256
 ↳ fee1) = _withdraw(liquidityBurned);
255 
256             _applyPerformanceFee(fee0, fee1);
257             (fee0, fee1) = _netPerformanceFees(fee0, fee1);
258             emit FeesEarned(fee0, fee1);
259 
```

```
260             amount0 =
261                 burn0 +
262                 FullMath.mulDiv(
263                     token0.balanceOf(address(this)) - burn0 -
 ↳ managerBalance0,
264                     burnAmount,
265                     totalSupply
266                 );
267             emit FeesEarned(fee0, fee1);
268
269             amount1 =
270                 burn1 +
271                 FullMath.mulDiv(
272                     token1.balanceOf(address(this)) - burn1 -
 ↳ managerBalance1,
273                     burnAmount,
274                     totalSupply
```

Listing 14: RangeProtocolVault.sol (Line 181)

```
171 function mint(
172         uint256 mintAmount
173     ) external override nonReentrant whenNotPaused returns (
 ↳ uint256 amount0, uint256 amount1) {
174         if (!mintStarted) revert VaultErrors.MintNotStarted();
175         if (mintAmount == 0) revert VaultErrors.InvalidMintAmount
 ↳ ();
176         uint256 totalSupply = totalSupply();
177         bool _inThePosition = inThePosition;
178         (uint160 sqrtRatioX96, , , , , , ) = pool.slot0();
179
180         if (totalSupply > 0) {
181             (uint256 amount0Current, uint256 amount1Current) =
 ↳ getUnderlyingBalances();
```

Listing 15: RangeProtocolVault.sol (Lines 629-630)

```
604 function _getUnderlyingBalances(
605         uint160 sqrtRatioX96,
606         int24 tick
607     ) internal view returns (uint256 amount0Current, uint256
 ↳ amount1Current) {
608         (
            uint128 liquidity,
            uint256 feeGrowthInside0Last,
            uint256 feeGrowthInside1Last,
            uint128 tokensOwed0,
            uint128 tokensOwed1
        ) = pool.positions(getPositionID());

        uint256 fee0;
        uint256 fee1;
        if (liquidity != 0) {
            (amount0Current, amount1Current) = LiquidityAmounts.getAmountsForLiquidity(
                sqrtRatioX96,
                lowerTick.getSqrtRatioAtTick(),
                upperTick.getSqrtRatioAtTick(),
                liquidity
            );
            fee0 = _feesEarned(true, feeGrowthInside0Last, tick, liquidity) + uint256(tokensOwed0);
            fee1 = _feesEarned(false, feeGrowthInside1Last, tick, liquidity) + uint256(tokensOwed1);
            (fee0, fee1) = _netPerformanceFees(fee0, fee1);
        }
        amount0Current += fee0 + token0.balanceOf(address(this)) - managerBalance0;
        amount1Current += fee1 + token1.balanceOf(address(this)) - managerBalance1;
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:H/Y:C/R:P/S:U (1.4)

Recommendation:

It is recommended to verify that the contract balance is greater than managerBalance0 and managerBalance1.

Remediation Plan:

ACKNOWLEDGED: The Range Protocol team partially solved this issue in commit f0777366 adding the following changes:

- The arithmetic operation in the burn() function that calculates **amount0** and **amount1**. The **managerBalance0** and **managerBalance1** values are subtracted only if they are lesser than the results of the subtractions of the contract balance and the values burn0 and burn1, thus preventing the underflow.

Listing 16: RangeProtocolVault.sol

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

- The arithmetic operation in the _getUnderlyingBalances() function, called by the mint() function, which calculates **amountCurrent0** and **amountCurrent1**. The **managerBalance0** and **managerBalance1** values are subtracted only if they are lesser than the results of the addition of the contract balance and the values **fee0** and **fee1** thus preventing the underflow.

Listing 17: RangeProtocolVault.sol

```
uint256 passiveBalance0 = fee0 + token0.balanceOf(address(this));
uint256 passiveBalance1 = fee1 + token1.balanceOf(address(this));
amount0Current += passiveBalance0 > managerBalance0
    ? passiveBalance0 - managerBalance0
    : passiveBalance0;
amount1Current += passiveBalance1 > managerBalance1
    ? passiveBalance1 - managerBalance1
    : passiveBalance1;
```

The Range Protocol team acknowledges the case in which the manager does not collect the fees when the balance is too low (excluding the corresponding manager fees) after burning, and the `addLiquidity()` function is called, which includes the fees in the liquidity, and so they are no longer available for collecting. Thus, the manager is not able to collect the fees with the `collectManager()` function at that time and has to wait for several operations to be performed.

RETESTING

The issue described in this section was brought to Halborn's attention by the Range Protocol team during the engagement.

5.1 RANGE01 - NEW POOL POSITIONS CAN BE OPENED BY VAULT MANAGER

Description:

The RangeProtocolVault contract is a vault solution that uses users' liquidity to serve a UniSwap V3 pool. The vault's manager can removeLiqudity() and addLiqudity() to the pool. However, to work with the pool, a new position must be opened within the range of valid ticks. The solution is meant to work with only one single position. The position is defined by lower and upper ticks. To start the vault, the manager must firstly set the ticks by means of updateTicks(). To update ticks, the manager must first remove all liquidity from the pool with the removeLiqudity() function. Internally, to work with the pool, almost all internal functions use saved tick values. However, the addLiqudity() function has two input parameters: newLowerTick and newUpperTick. Therefore, the vault manager can provide invalid newLowerTick and newUpperTick parameters mistakenly (different from set with the updateTicks() function), and as a result new position would be opened in the pool.

Code Location:

Listing 18: RangeProtocolVault.sol (Lines 361,362,377-383)

```
360 function addLiquidity(
361         int24 newLowerTick,
362         int24 newUpperTick,
363         uint256 amount0,
364         uint256 amount1
365     ) external override onlyManager returns (uint256
  ↳ remainingAmount0, uint256 remainingAmount1) {
366         _validateTicks(newLowerTick, newUpperTick);
367         (uint160 sqrtRatioX96, , , , , , ) = pool.slot0();
```

```
        uint128 baseLiquidity = LiquidityAmounts.getLiquidityForAmounts(
            sqrtRatioX96,
            newLowerTick.getSqrtRatioAtTick(),
            newUpperTick.getSqrtRatioAtTick(),
            amount0,
            amount1
        );

        if (baseLiquidity > 0) {
            (uint256 amountDeposited0, uint256 amountDeposited1) = pool.mint(
                address(this),
                newLowerTick,
                newUpperTick,
                baseLiquidity,
                ""
            );
            // Should return remaining token number for swap
            remainingAmount0 = amount0 - amountDeposited0;
            remainingAmount1 = amount1 - amountDeposited1;
            if (lowerTick != newLowerTick || upperTick != newUpperTick) {
                lowerTick = newLowerTick;
                upperTick = newUpperTick;
                emit TicksSet(newLowerTick, newUpperTick);
            }

            emit LiquidityAdded(
                baseLiquidity,
                newLowerTick,
                newUpperTick,
                amountDeposited0,
                amountDeposited1
            );
        }
        // This check is added to not update inThePosition state in case manager decides to add liquidity in smaller chunks.
        if (!inThePosition) {
            inThePosition = true;
            emit InThePositionStatusSet(true);
        }
    }
```

BVSS::

AO:S/AC:L/AX:L/C:N/I:H/A:H/D:N/Y:H/R:P/S:U (1.1 - Informational)

Recommendation:

It is recommended to prevent adding liquidity and opening new position based on input parameters to the addLiquidity() function.

Remediation Plan:

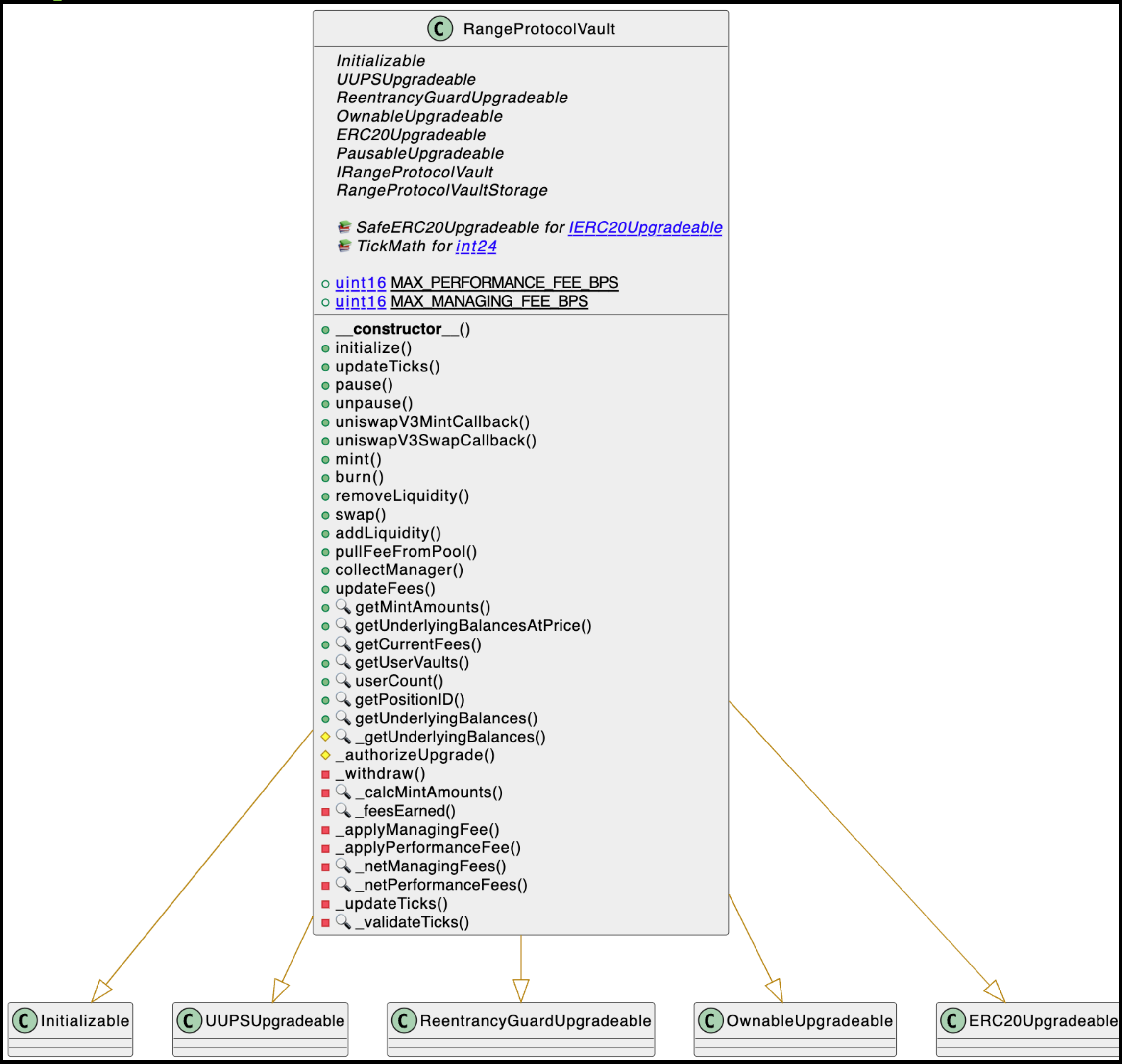
SOLVED: The Range Protocol team identified this issue and solved it by preventing calling the addLiquidity() function more than once without a prior call to the removeLiquidity() function in commit a883d28c.

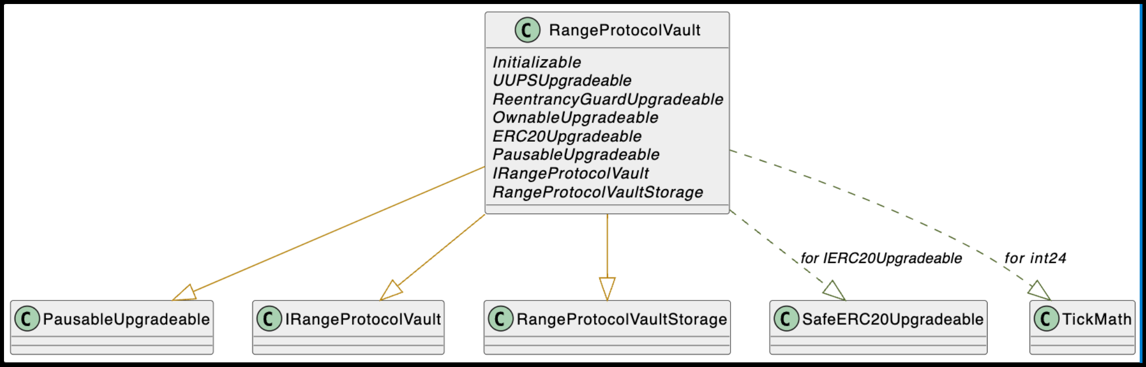
CONTRACT UPGRADABILITY

6.1 Solution Structure

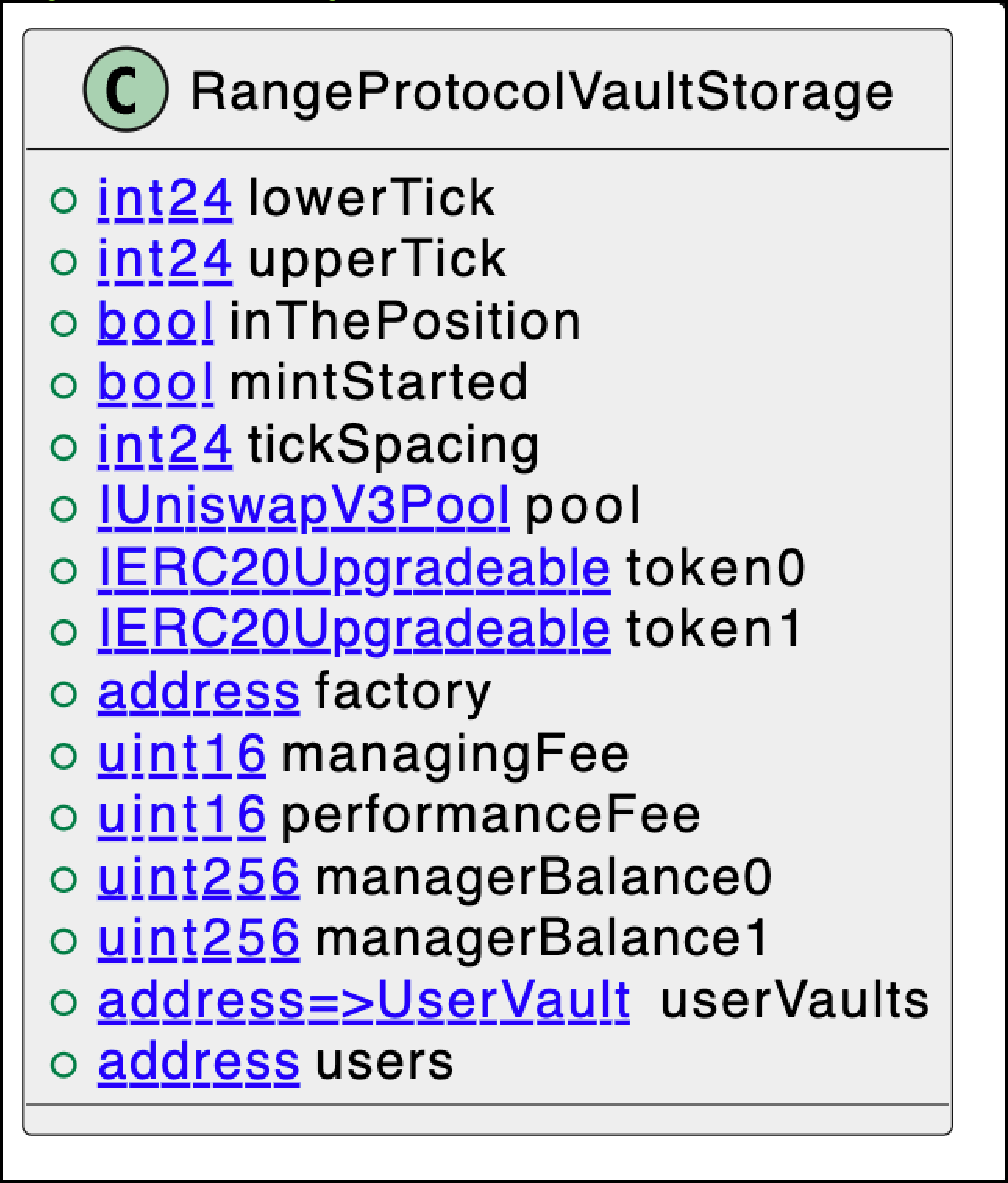
The team at Halborn analyzed the structure of the smart contracts in scope to make sure all future upgrades are secure. The Range Protocol team decided to use the UUPSUpgradeable pattern for the RangeProtocolVault solution.

RangeProtocolVault.sol



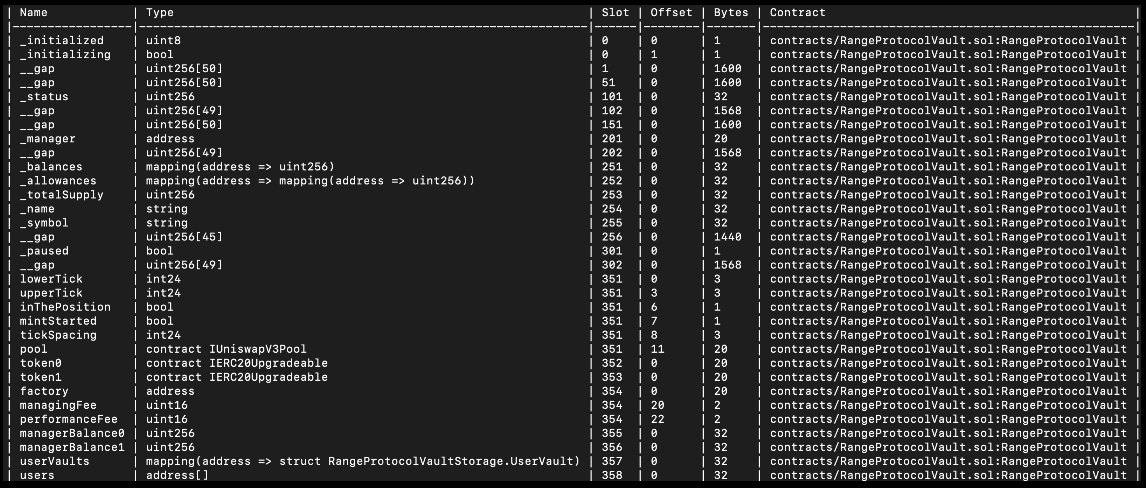

RangeProtocolVault
Initializable
UUPSUpgradeable
ReentrancyGuardUpgradeable
OwnableUpgradeable
ERC20Upgradeable
PausableUpgradeable
IRangeProtocolVault
RangeProtocolVaultStorage
for IERC20Upgradeable
for int24
PausableUpgradeable
IRangeProtocolVault
RangeProtocolVaultStorage
SafeERC20Upgradeable
TickMath

RangeProtocolVaultStorage.sol



6.2 Storage

No possibility of storage collision between the proxy and implementation was identified. The Range Protocol team is using the standard ERC1967Proxy along with the UUPSUpgradeable.



```
| Name            | Type                                                           | Slot | Offset | Bytes | Contract                                            |
|-----------------|----------------------------------------------------------------|------|--------|-------|-----------------------------------------------------|
| _initialized    | uint8                                                          | 0    | 0      | 1     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _initializing   | bool                                                           | 0    | 1      | 1     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[50]                                                    | 1    | 0      | 1600  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[50]                                                    | 51   | 0      | 1600  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _status         | uint256                                                        | 101  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[49]                                                    | 102  | 0      | 1568  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[50]                                                    | 151  | 0      | 1600  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _manager        | address                                                        | 201  | 0      | 20    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[49]                                                    | 202  | 0      | 1568  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _balances       | mapping(address => uint256)                                    | 251  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _allowances     | mapping(address => mapping(address => uint256))                | 252  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _totalSupply    | uint256                                                        | 253  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _name           | string                                                         | 254  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _symbol         | string                                                         | 255  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[45]                                                    | 256  | 0      | 1440  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| _paused         | bool                                                           | 301  | 0      | 1     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| __gap           | uint256[49]                                                    | 302  | 0      | 1568  | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| lowerTick       | int24                                                          | 351  | 0      | 3     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| upperTick       | int24                                                          | 351  | 3      | 3     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| inThePosition   | bool                                                           | 351  | 6      | 1     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| mintStarted     | bool                                                           | 351  | 7      | 1     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| tickSpacing     | int24                                                          | 351  | 8      | 3     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| pool            | contract IUniswapV3Pool                                        | 351  | 11     | 20    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| token0          | contract IERC20Upgradeable                                     | 352  | 0      | 20    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| token1          | contract IERC20Upgradeable                                     | 353  | 0      | 20    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| factory         | address                                                        | 354  | 0      | 20    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| managingFee     | uint16                                                         | 354  | 20     | 2     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| performanceFee  | uint16                                                         | 354  | 22     | 2     | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| managerBalance0 | uint256                                                        | 355  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| managerBalance1 | uint256                                                        | 356  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| userVaults      | mapping(address => struct RangeProtocolVaultStorage.UserVault) | 357  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
| users           | address[]                                                      | 358  | 0      | 32    | contracts/RangeProtocolVault.sol:RangeProtocolVault |
```

6.3 Initialization

Every initialization function is correctly protected with the initializer modifier, preventing any possible re-initialization:

Listing 19: RangeProtocolVault.sol (Line 74)

```
70     function initialize(
71         address _pool,
72         int24 _tickSpacing,
73         bytes memory data
74     ) external override initializer {
75         (address manager, string memory _name, string memory
 ↳ _symbol) = abi.decode(
76             data,
77             (address, string, string)
78         );
79 
80         __UUPSUpgradeable_init();
81         __ReentrancyGuard_init();
```

```
        __Ownable_init();
        __ERC20_init(_name, _symbol);
        __Pausable_init();

        _transferOwnership(manager);

        pool = IUniswapV3Pool(_pool);
        token0 = IERC20Upgradeable(pool.token0());
        token1 = IERC20Upgradeable(pool.token1());
        tickSpacing = _tickSpacing;
        factory = msg.sender;

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

The RangeProtocolVaultStorage contract is the only custom parent contract for the RangeProtocolVault and it has no initializer. Other third-party parent contracts are correctly initialized.

RangeProtocolVault
- UUPSUpgradeable [X]
- ReentrancyGuardUpgradeable [X]
- OwnableUpgradeable [X]
- ERC20Upgradeable [X]
- PausableUpgradeable [X]

All relevant contracts implement constructor with _disableInitializers():

Listing 20: RangeProtocolVault.sol

```
    constructor() {
        _disableInitializers();
    }
```

6.4 Deployment

The RangeProtocolVault is being deployed and initialized with a single transaction by means of the 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();
    if (tokenA == tokenB) revert();
    address token0 = tokenA < tokenB ? tokenA : tokenB;
    if (token0 == address(0x0)) revert("token cannot be a zero address");

    int24 tickSpacing = IUniswapV3Factory(factory).feeAmountTickSpacing(fee);
    vault = address(
        new ERC1967Proxy(
            implementation,
            abi.encodeWithSelector(INIT_SELECTOR, pool, tickSpacing, data)
        )
    );
    _vaultsList.push(vault);
}
```

AUTOMATED TESTING

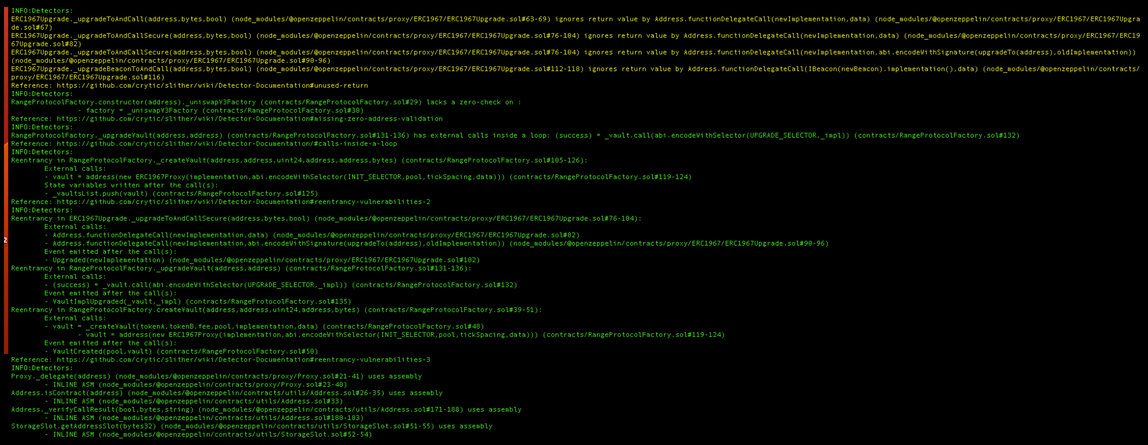
7.1 STATIC ANALYSIS REPORT

Description:

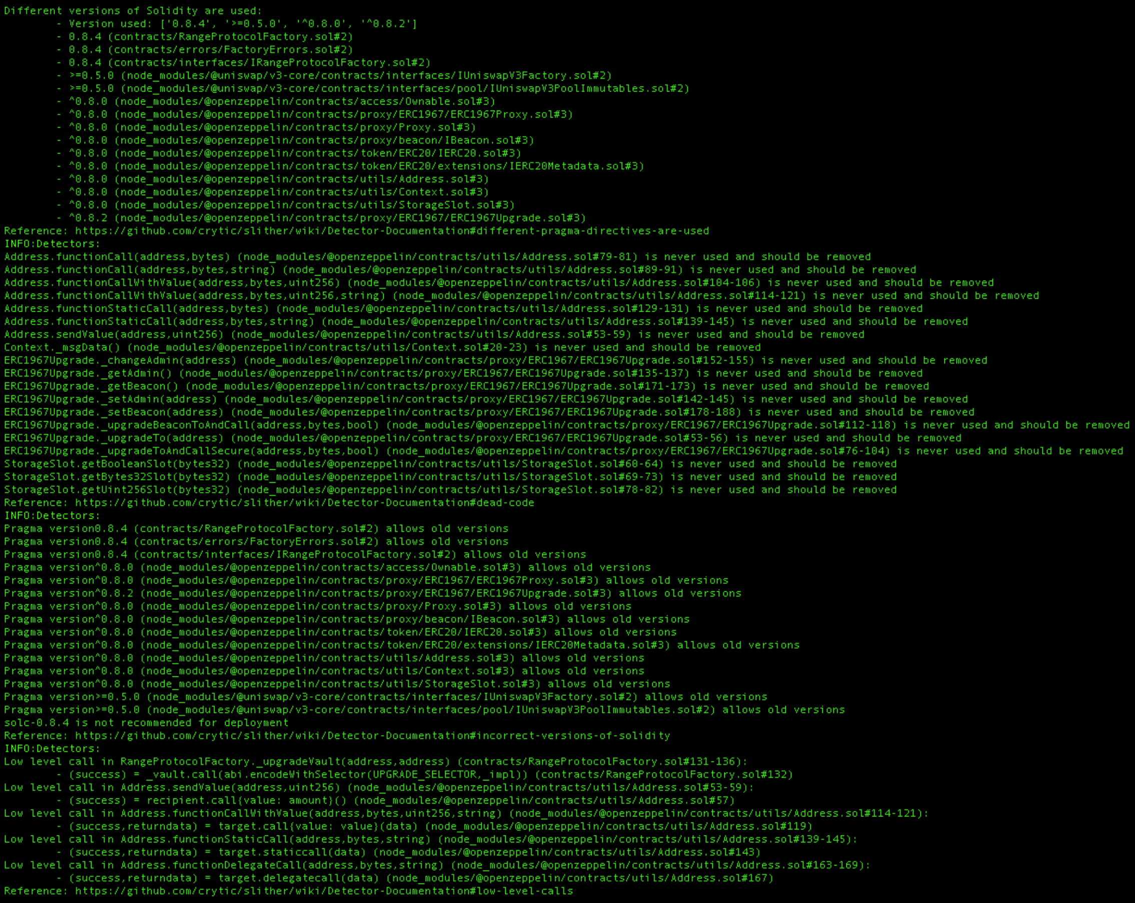
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

RangeProtocolFactory

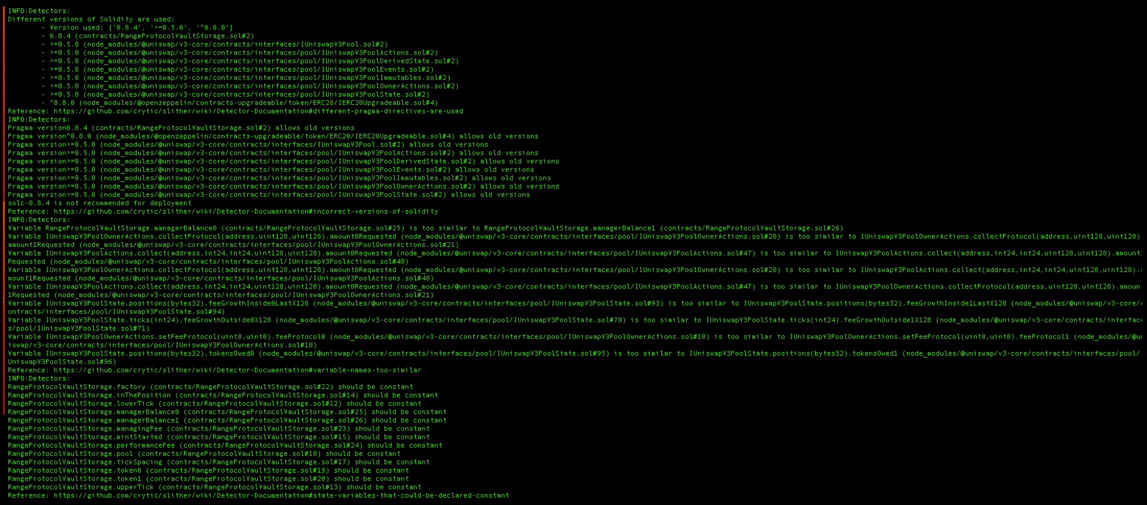


```
INFO:Detectors:
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#63-69) ignores return value by Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgra
de.sol#67)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#76-104) ignores return value by Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC19
67Upgrade.sol#82)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#76-104) ignores return value by Address.functionDelegateCall(newImplementation,abi.encodeWithSignature(upgradeTo(address),oldImplementation))
(node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#90-96)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#112-118) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (node_modules/@openzeppelin/contracts/
proxy/ERC1967/ERC1967Upgrade.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
RangeProtocolFactory.constructor(address)._uniswapV3Factory (contracts/RangeProtocolFactory.sol#29) lacks a zero-check on :
                - factory = _uniswapV3Factory (contracts/RangeProtocolFactory.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
RangeProtocolFactory._upgradeVault(address,address) (contracts/RangeProtocolFactory.sol#131-136) has external calls inside a loop: (success) = _vault.call(abi.encodeWithSelector(UPGRADE_SELECTOR,_impl)) (contracts/RangeProtocolFactory.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in RangeProtocolFactory._createVault(address,address,uint24,address,address,bytes) (contracts/RangeProtocolFactory.sol#105-126):
        External calls:
        - vault = address(new ERC1967Proxy(implementation,abi.encodeWithSelector(INIT_SELECTOR,pool,tickSpacing,data))) (contracts/RangeProtocolFactory.sol#119-124)
        State variables written after the call(s):
        - _vaultsList.push(vault) (contracts/RangeProtocolFactory.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#76-104):
        External calls:
        - Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#82)
        - Address.functionDelegateCall(newImplementation,abi.encodeWithSignature(upgradeTo(address),oldImplementation)) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#90-96)
        Event emitted after the call(s):
        - Upgraded(newImplementation) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#102)
Reentrancy in RangeProtocolFactory._upgradeVault(address,address) (contracts/RangeProtocolFactory.sol#131-136):
        External calls:
        - (success) = _vault.call(abi.encodeWithSelector(UPGRADE_SELECTOR,_impl)) (contracts/RangeProtocolFactory.sol#132)
        Event emitted after the call(s):
        - VaultImplUpgraded(_vault,_impl) (contracts/RangeProtocolFactory.sol#135)
Reentrancy in RangeProtocolFactory.createVault(address,address,uint24,address,bytes) (contracts/RangeProtocolFactory.sol#39-51):
        External calls:
        - vault = _createVault(tokenA,tokenB,fee,pool,implementation,data) (contracts/RangeProtocolFactory.sol#48)
                - vault = address(new ERC1967Proxy(implementation,abi.encodeWithSelector(INIT_SELECTOR,pool,tickSpacing,data))) (contracts/RangeProtocolFactory.sol#119-124)
        Event emitted after the call(s):
        - VaultCreated(pool,vault) (contracts/RangeProtocolFactory.sol#50)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Proxy._delegate(address) (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#21-41) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#23-40)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#171-188) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#180-183)
StorageSlot.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#51-55) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#52-54)
```



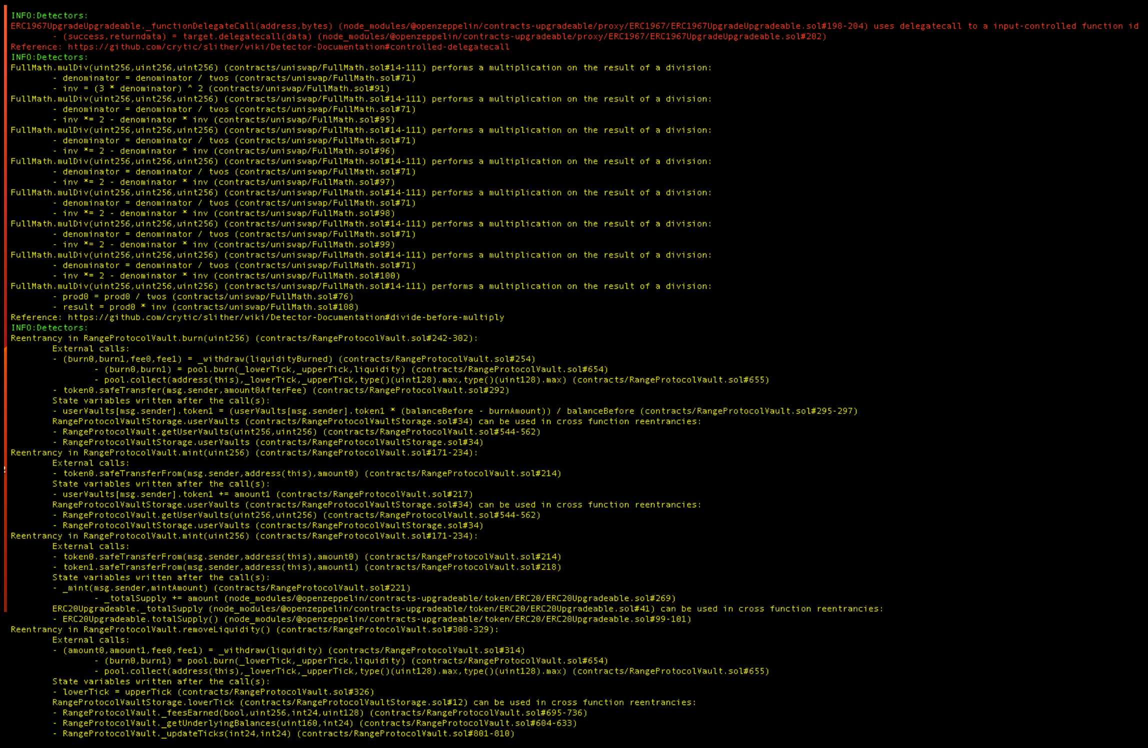
```
Different versions of Solidity are used:
        - Version used: ['0.8.4', '>=0.5.0', '^0.8.0', '^0.8.2']
        - 0.8.4 (contracts/RangeProtocolFactory.sol#2)
        - 0.8.4 (contracts/errors/FactoryErrors.sol#2)
        - 0.8.4 (contracts/interfaces/IRangeProtocolFactory.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolImmutables.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/proxy/beacon/IBeacon.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#3)
        - ^0.8.2 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#79-81) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#89-91) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#104-106) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#129-131) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-23) is never used and should be removed
ERC1967Upgrade._changeAdmin(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#152-155) is never used and should be removed
ERC1967Upgrade._getAdmin() (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#135-137) is never used and should be removed
ERC1967Upgrade._getBeacon() (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#171-173) is never used and should be removed
ERC1967Upgrade._setAdmin(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#142-145) is never used and should be removed
ERC1967Upgrade._setBeacon(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#178-188) is never used and should be removed
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#112-118) is never used and should be removed
ERC1967Upgrade._upgradeTo(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#53-56) is never used and should be removed
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#76-104) is never used and should be removed
StorageSlot.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#60-64) is never used and should be removed
StorageSlot.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#69-73) is never used and should be removed
StorageSlot.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#78-82) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (contracts/RangeProtocolFactory.sol#2) allows old versions
Pragma version0.8.4 (contracts/errors/FactoryErrors.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IRangeProtocolFactory.sol#2) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol#3) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/proxy/beacon/IBeacon.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#3) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolImmutables.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in RangeProtocolFactory._upgradeVault(address,address) (contracts/RangeProtocolFactory.sol#131-136):
        - (success) = _vault.call(abi.encodeWithSelector(UPGRADE_SELECTOR,_impl)) (contracts/RangeProtocolFactory.sol#132)
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

RangeProtocolVaultStorage

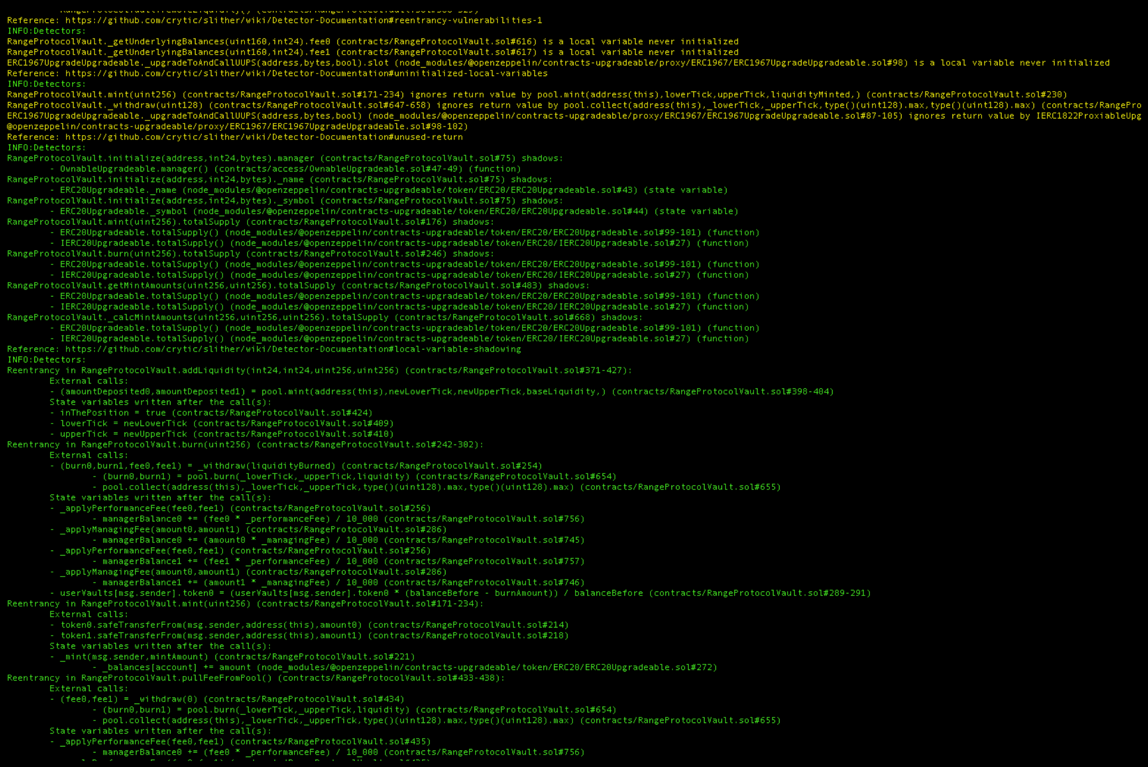


```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.4', '>=0.5.0', '^0.8.0']
        - 0.8.4 (contracts/RangeProtocolVaultStorage.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolDerivedState.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolEvents.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolImmutables.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#2)
        - >=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolState.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version0.8.4 (contracts/RangeProtocolVaultStorage.sol#2) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolDerivedState.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolEvents.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolImmutables.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#2) allows old versions
Pragma version>=0.5.0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolState.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable RangeProtocolVaultStorage.managerBalance0 (contracts/RangeProtocolVaultStorage.sol#25) is too similar to RangeProtocolVaultStorage.managerBalance1 (contracts/RangeProtocolVaultStorage.sol#26)
Variable IUniswapV3PoolOwnerActions.collectProtocol(address,uint128,uint128).amount0Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#20) is too similar to IUniswapV3PoolOwnerActions.collectProtocol(address,uint128,uint128)
amount1Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#21)
Variable IUniswapV3PoolActions.collect(address,int24,int24,uint128,uint128).amount0Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#47) is too similar to IUniswapV3PoolActions.collect(address,int24,int24,uint128,uint128).amount
Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#48)
Variable IUniswapV3PoolOwnerActions.collectProtocol(address,uint128,uint128).amount0Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#20) is too similar to IUniswapV3PoolActions.collect(address,int24,int24,uint128,uint128).
mount1Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#48)
Variable IUniswapV3PoolActions.collect(address,int24,int24,uint128,uint128).amount0Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolActions.sol#47) is too similar to IUniswapV3PoolOwnerActions.collectProtocol(address,uint128,uint128).amoun
1Requested (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#21)
Variable IUniswapV3PoolState.positions(bytes32).feeGrowthInside0LastX128 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolState.sol#93) is too similar to IUniswapV3PoolState.positions(bytes32).feeGrowthInside1LastX128 (node_modules/@uniswap/v3-core/
ontracts/interfaces/pool/IUniswapV3PoolState.sol#94)
Variable IUniswapV3PoolState.ticks(int24).feeGrowthOutside0X128 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolState.sol#70) is too similar to IUniswapV3PoolState.ticks(int24).feeGrowthOutside1X128 (node_modules/@uniswap/v3-core/contracts/interfac
s/pool/IUniswapV3PoolState.sol#71)
Variable IUniswapV3PoolOwnerActions.setFeeProtocol(uint8,uint8).feeProtocol0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#10) is too similar to IUniswapV3PoolOwnerActions.setFeeProtocol(uint8,uint8).feeProtocol1 (node_modules/@u
iswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol#10)
Variable IUniswapV3PoolState.positions(bytes32).tokensOwed0 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/IUniswapV3PoolState.sol#95) is too similar to IUniswapV3PoolState.positions(bytes32).tokensOwed1 (node_modules/@uniswap/v3-core/contracts/interfaces/pool/
UniswapV3PoolState.sol#96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
RangeProtocolVaultStorage.factory (contracts/RangeProtocolVaultStorage.sol#22) should be constant
RangeProtocolVaultStorage.inThePosition (contracts/RangeProtocolVaultStorage.sol#14) should be constant
RangeProtocolVaultStorage.lowerTick (contracts/RangeProtocolVaultStorage.sol#12) should be constant
RangeProtocolVaultStorage.managerBalance0 (contracts/RangeProtocolVaultStorage.sol#25) should be constant
RangeProtocolVaultStorage.managerBalance1 (contracts/RangeProtocolVaultStorage.sol#26) should be constant
RangeProtocolVaultStorage.managingFee (contracts/RangeProtocolVaultStorage.sol#23) should be constant
RangeProtocolVaultStorage.mintStarted (contracts/RangeProtocolVaultStorage.sol#15) should be constant
RangeProtocolVaultStorage.performanceFee (contracts/RangeProtocolVaultStorage.sol#24) should be constant
RangeProtocolVaultStorage.pool (contracts/RangeProtocolVaultStorage.sol#18) should be constant
RangeProtocolVaultStorage.tickSpacing (contracts/RangeProtocolVaultStorage.sol#17) should be constant
RangeProtocolVaultStorage.token0 (contracts/RangeProtocolVaultStorage.sol#19) should be constant
RangeProtocolVaultStorage.token1 (contracts/RangeProtocolVaultStorage.sol#20) should be constant
RangeProtocolVaultStorage.upperTick (contracts/RangeProtocolVaultStorage.sol#13) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

RangeProtocolVault



```
INFO:Detectors:
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function id
	- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv = (3 * denominator) ^ 2 (contracts/uniswap/FullMath.sol#91)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#95)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#96)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#97)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#98)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#99)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- denominator = denominator / twos (contracts/uniswap/FullMath.sol#71)
	- inv *= 2 - denominator * inv (contracts/uniswap/FullMath.sol#100)
FullMath.mulDiv(uint256,uint256,uint256) (contracts/uniswap/FullMath.sol#14-111) performs a multiplication on the result of a division:
	- prod0 = prod0 / twos (contracts/uniswap/FullMath.sol#76)
	- result = prod0 * inv (contracts/uniswap/FullMath.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in RangeProtocolVault.burn(uint256) (contracts/RangeProtocolVault.sol#242-302):
	External calls:
	- (burn0,burn1,fee0,fee1) = _withdraw(liquidityBurned) (contracts/RangeProtocolVault.sol#254)
		- (burn0,burn1) = pool.burn(_lowerTick,_upperTick,liquidity) (contracts/RangeProtocolVault.sol#654)
		- pool.collect(address(this),_lowerTick,_upperTick,type()(uint128).max,type()(uint128).max) (contracts/RangeProtocolVault.sol#655)
	- token0.safeTransfer(msg.sender,amount0AfterFee) (contracts/RangeProtocolVault.sol#292)
	State variables written after the call(s):
	- userVaults[msg.sender].token1 = (userVaults[msg.sender].token1 * (balanceBefore - burnAmount)) / balanceBefore (contracts/RangeProtocolVault.sol#295-297)
	RangeProtocolVaultStorage.userVaults (contracts/RangeProtocolVaultStorage.sol#34) can be used in cross function reentrancies:
	- RangeProtocolVault.getUserVaults(uint256,uint256) (contracts/RangeProtocolVault.sol#544-562)
	- RangeProtocolVaultStorage.userVaults (contracts/RangeProtocolVaultStorage.sol#34)
Reentrancy in RangeProtocolVault.mint(uint256) (contracts/RangeProtocolVault.sol#171-234):
	External calls:
	- token0.safeTransferFrom(msg.sender,address(this),amount0) (contracts/RangeProtocolVault.sol#214)
	State variables written after the call(s):
	- userVaults[msg.sender].token1 += amount1 (contracts/RangeProtocolVault.sol#217)
	RangeProtocolVaultStorage.userVaults (contracts/RangeProtocolVaultStorage.sol#34) can be used in cross function reentrancies:
	- RangeProtocolVault.getUserVaults(uint256,uint256) (contracts/RangeProtocolVault.sol#544-562)
	- RangeProtocolVaultStorage.userVaults (contracts/RangeProtocolVaultStorage.sol#34)
Reentrancy in RangeProtocolVault.mint(uint256) (contracts/RangeProtocolVault.sol#171-234):
	External calls:
	- token0.safeTransferFrom(msg.sender,address(this),amount0) (contracts/RangeProtocolVault.sol#214)
	- token1.safeTransferFrom(msg.sender,address(this),amount1) (contracts/RangeProtocolVault.sol#218)
	State variables written after the call(s):
	- _mint(msg.sender,mintAmount) (contracts/RangeProtocolVault.sol#221)
		- _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
	ERC20Upgradeable._totalSupply (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#41) can be used in cross function reentrancies:
	- ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101)
Reentrancy in RangeProtocolVault.removeLiquidity() (contracts/RangeProtocolVault.sol#308-329):
	External calls:
	- (amount0,amount1,fee0,fee1) = _withdraw(liquidity) (contracts/RangeProtocolVault.sol#314)
		- (burn0,burn1) = pool.burn(_lowerTick,_upperTick,liquidity) (contracts/RangeProtocolVault.sol#654)
		- pool.collect(address(this),_lowerTick,_upperTick,type()(uint128).max,type()(uint128).max) (contracts/RangeProtocolVault.sol#655)
	State variables written after the call(s):
	- lowerTick = upperTick (contracts/RangeProtocolVault.sol#326)
	RangeProtocolVaultStorage.lowerTick (contracts/RangeProtocolVaultStorage.sol#12) can be used in cross function reentrancies:
	- RangeProtocolVault._feesEarned(bool,uint256,int24,uint128) (contracts/RangeProtocolVault.sol#695-736)
	- RangeProtocolVault._getUnderlyingBalances(uint160,int24) (contracts/RangeProtocolVault.sol#604-633)
	- RangeProtocolVault._updateTicks(int24,int24) (contracts/RangeProtocolVault.sol#801-810)
```



```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
RangeProtocolVault._getUnderlyingBalances(uint160,int24).fee0 (contracts/RangeProtocolVault.sol#616) is a local variable never initialized
RangeProtocolVault._getUnderlyingBalances(uint160,int24).fee1 (contracts/RangeProtocolVault.sol#617) is a local variable never initialized
ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
RangeProtocolVault.mint(uint256) (contracts/RangeProtocolVault.sol#171-234) ignores return value by pool.mint(address(this),lowerTick,upperTick,liquidityMinted,) (contracts/RangeProtocolVault.sol#230)
RangeProtocolVault._withdraw(uint128) (contracts/RangeProtocolVault.sol#647-658) ignores return value by pool.collect(address(this),_lowerTick,_upperTick,type()(uint128).max,type()(uint128).max) (contracts/RangePro
ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpg
@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
RangeProtocolVault.initialize(address,int24,bytes).manager (contracts/RangeProtocolVault.sol#75) shadows:
	- OwnableUpgradeable.manager() (contracts/access/OwnableUpgradeable.sol#47-49) (function)
RangeProtocolVault.initialize(address,int24,bytes)._name (contracts/RangeProtocolVault.sol#75) shadows:
	- ERC20Upgradeable._name (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#43) (state variable)
RangeProtocolVault.initialize(address,int24,bytes)._symbol (contracts/RangeProtocolVault.sol#75) shadows:
	- ERC20Upgradeable._symbol (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#44) (state variable)
RangeProtocolVault.mint(uint256).totalSupply (contracts/RangeProtocolVault.sol#176) shadows:
	- ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101) (function)
	- IERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#27) (function)
RangeProtocolVault.burn(uint256).totalSupply (contracts/RangeProtocolVault.sol#246) shadows:
	- ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101) (function)
	- IERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#27) (function)
RangeProtocolVault.getMintAmounts(uint256,uint256).totalSupply (contracts/RangeProtocolVault.sol#483) shadows:
	- ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101) (function)
	- IERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#27) (function)
RangeProtocolVault._calcMintAmounts(uint256,uint256,uint256).totalSupply (contracts/RangeProtocolVault.sol#668) shadows:
	- ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101) (function)
	- IERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#27) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in RangeProtocolVault.addLiquidity(int24,int24,uint256,uint256) (contracts/RangeProtocolVault.sol#371-427):
	External calls:
	- (amountDeposited0,amountDeposited1) = pool.mint(address(this),newLowerTick,newUpperTick,baseLiquidity,) (contracts/RangeProtocolVault.sol#398-404)
	State variables written after the call(s):
	- inThePosition = true (contracts/RangeProtocolVault.sol#424)
	- lowerTick = newLowerTick (contracts/RangeProtocolVault.sol#409)
	- upperTick = newUpperTick (contracts/RangeProtocolVault.sol#410)
Reentrancy in RangeProtocolVault.burn(uint256) (contracts/RangeProtocolVault.sol#242-302):
	External calls:
	- (burn0,burn1,fee0,fee1) = _withdraw(liquidityBurned) (contracts/RangeProtocolVault.sol#254)
		- (burn0,burn1) = pool.burn(_lowerTick,_upperTick,liquidity) (contracts/RangeProtocolVault.sol#654)
		- pool.collect(address(this),_lowerTick,_upperTick,type()(uint128).max,type()(uint128).max) (contracts/RangeProtocolVault.sol#655)
	State variables written after the call(s):
	- _applyPerformanceFee(fee0,fee1) (contracts/RangeProtocolVault.sol#256)
		- managerBalance0 += (fee0 * _performanceFee) / 10_000 (contracts/RangeProtocolVault.sol#756)
	- _applyManagingFee(amount0,amount1) (contracts/RangeProtocolVault.sol#286)
		- managerBalance0 += (amount0 * _managingFee) / 10_000 (contracts/RangeProtocolVault.sol#745)
	- _applyPerformanceFee(fee0,fee1) (contracts/RangeProtocolVault.sol#256)
		- managerBalance1 += (fee1 * _performanceFee) / 10_000 (contracts/RangeProtocolVault.sol#757)
	- _applyManagingFee(amount0,amount1) (contracts/RangeProtocolVault.sol#286)
		- managerBalance1 += (amount1 * _managingFee) / 10_000 (contracts/RangeProtocolVault.sol#746)
	- userVaults[msg.sender].token0 = (userVaults[msg.sender].token0 * (balanceBefore - burnAmount)) / balanceBefore (contracts/RangeProtocolVault.sol#289-291)
Reentrancy in RangeProtocolVault.mint(uint256) (contracts/RangeProtocolVault.sol#171-234):
	External calls:
	- token0.safeTransferFrom(msg.sender,address(this),amount0) (contracts/RangeProtocolVault.sol#214)
	- token1.safeTransferFrom(msg.sender,address(this),amount1) (contracts/RangeProtocolVault.sol#218)
	State variables written after the call(s):
	- _mint(msg.sender,mintAmount) (contracts/RangeProtocolVault.sol#221)
		- _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#272)
Reentrancy in RangeProtocolVault.pullFeeFromPool() (contracts/RangeProtocolVault.sol#433-438):
	External calls:
	- (fee0,fee1) = _withdraw(0) (contracts/RangeProtocolVault.sol#434)
		- (burn0,burn1) = pool.burn(_lowerTick,_upperTick,liquidity) (contracts/RangeProtocolVault.sol#654)
		- pool.collect(address(this),_lowerTick,_upperTick,type()(uint128).max,type()(uint128).max) (contracts/RangeProtocolVault.sol#655)
	State variables written after the call(s):
	- _applyPerformanceFee(fee0,fee1) (contracts/RangeProtocolVault.sol#435)
		- managerBalance0 += (fee0 * _performanceFee) / 10_000 (contracts/RangeProtocolVault.sol#756)
```

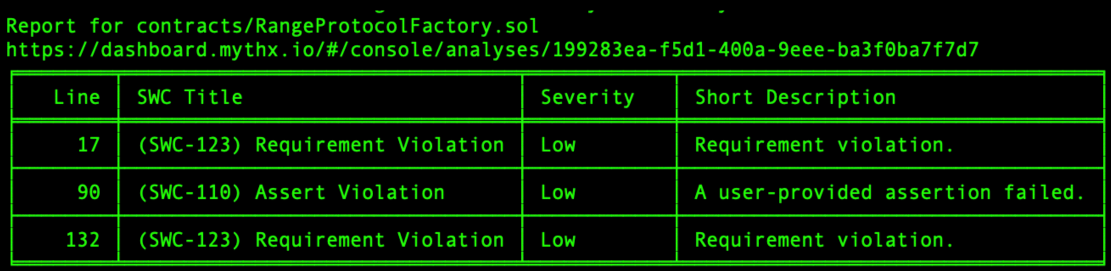
All the issues flagged by Slither were found to be either false positives or issues already reported.

7.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:



```
Report for contracts/RangeProtocolFactory.sol
https://dashboard.mythx.io/#/console/analyses/199283ea-f5d1-400a-9eee-ba3f0ba7f7d7
```

Line	SWC Title	Severity	Short Description
17	(SWC-123) Requirement Violation	Low	Requirement violation.
90	(SWC-110) Assert Violation	Low	A user-provided assertion failed.
132	(SWC-123) Requirement Violation	Low	Requirement violation.

All the issues flagged by MythX were found to be either false positives or issues already reported.

THANK YOU FOR CHOOSING

HALBORN