



StakeWise – Protocol V3

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 3rd, 2023 – April 28th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY	4
CONTACTS	5
1 EXECUTIVE OVERVIEW	6
1.1 INTRODUCTION	7
1.2 AUDIT SUMMARY	7
1.3 TEST APPROACH & METHODOLOGY	8
2 RISK METHODOLOGY	9
2.1 Exploitability	10
2.2 Impact	11
2.3 Severity Coefficient	13
2.4 SCOPE	15
3 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	17
4 FINDINGS & TECH DETAILS	18
4.1 (HAL-01) PRIVATE VAULT WHITELIST DEPOSIT BYPASS POSSIBLE - HIGH(7.0)	20
Description	20
Code Location	20
Proof of Concept	21
BVSS	22
Recommendation	22
Remediation Plan	22
4.2 (HAL-02) LACK OF EMERGENCY-STOP PATTERN - LOW(3.6)	23
Description	23
BVSS	23

Recommendation	23
Remediation Plan	23
4.3 (HAL-03) LACK OF ZERO ADDRESS CHECK FOR RECEIVER - LOW(3.4)	24
Description	24
Code Location	24
BVSS	26
Recommendation	26
Remediation Plan	27
4.4 (HAL-04) VAULTADMIN LACKS OWNERSHIP-TRANSFER PATTERN - LOW(3.0)	28
Description	28
BVSS	28
Recommendation	28
Remediation Plan	28
4.5 (HAL-05) INCREASEAPPROVE AND DECREASEAPPROVE CAN BE INTRODUCED - INFORMATIONAL(0.0)	29
Description	29
BVSS	29
Recommendation	29
Remediation Plan	29
5 MANUAL TESTING	30
6 CONTRACT UPGRADABILITY	36
6.1 Solution Structure	37
6.2 Storage	49
6.3 Initialization	50

6.4 Deployment	52
7 AUTOMATED TESTING	54
7.1 STATIC ANALYSIS REPORT	56
Description	56
Results	56
7.2 AUTOMATED SECURITY SCAN	60
Description	60
Results	60

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/03/2023	Grzegorz Trawinski
0.2	Document Updates	04/20/2023	Grzegorz Trawinski
0.3	Final Draft	04/27/2023	Grzegorz Trawinski
0.4	Draft Review	04/27/2023	Ataberk Yavuzer
0.5	Draft Review	04/27/2023	Piotr Cielas
0.6	Draft Review	04/28/2023	Gabi Urrutia
1.0	Remediation Plan	05/04/2023	Grzegorz Trawinski
1.1	Remediation Plan Review	05/04/2023	Ataberk Yavuzer
1.2	Remediation Plan Review	05/04/2023	Piotr Cielas
1.3	Remediation Plan Review	05/05/2023	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

StakeWise made ETH staking effortless by providing resilient infrastructure and yield strategies. StakeWise's clients can participate in Ethereum's Proof-of-Stake consensus mechanism (staking) and receive ETH rewards in return.

StakeWise engaged [Halborn](#) to conduct a security audit on their smart contracts beginning on April 3rd, 2023 and ending on April 28th, 2023. The security assessment was scoped to the smart contracts provided in the [v3-core](#) 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 4 weeks for the engagement and assigned 1 full-time security engineer to audit the security of the smart contracts in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

- Identify potential security issues within the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by StakeWise. The main one is related to the private vault deposit bypass.

On May 4th, 2023, Halborn team received the updated code repository. HAL-01, HAL-03 and HAL-05 findings were addressed and marked as [SOLVED](#). HAL-02 and HAL-04 findings were not resolved and marked as [RISK ACCEPTED](#), as this functionality works as intended along with StakeWise business

rules. Additionally, StakeWise decided to remove the `withdraw()` and `whitelist` features.

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
- 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 analysis (Remix IDE, [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. Project Name

- Repository: `v3-core`
- Commit ID: `7603afff540c75e4a2d4a26a891dcd2a214c1d6b`
- Smart contracts in scope:
 - `libraries/ExitQueue.sol`
 - `vaults/modules/VaultMev.sol`
 - `vaults/modules/VaultValidators.sol`
 - `vaults/modules/VaultWhitelist.sol`
 - `vaults/modules/VaultImmutables.sol`
 - `vaults/modules/VaultToken.sol`
 - `vaults/modules/VaultAdmin.sol`
 - `vaults/modules/VaultEthStaking.sol`
 - `vaults/modules/VaultFee.sol`
 - `vaults/modules/VaultVersion.sol`
 - `vaults/modules/VaultEnterExit.sol`
 - `vaults/modules/VaultState.sol`
 - `vaults/VaultsRegistry.sol`
 - `vaults/ethereum/EthPrivateVault.sol`
 - `vaults/ethereum/EthVaultFactory.sol`
 - `vaults/ethereum/EthVault.sol`
 - `vaults/ethereum/mev/SharedMevEscrow.sol`
 - `vaults/ethereum/mev/OwnMevEscrow.sol`
 - `keeper/KeeperValidators.sol`
 - `keeper/KeeperRewards.sol`
 - `keeper/Oracles.sol`
 - `keeper/Keeper.sol`
 - `base/ERC20Upgradeable.sol`
 - `base/Multicall.sol`
 - `base/Versioned.sol`
 - `interfaces/IOracles.sol`
 - `interfaces/IVaultWhitelist.sol`

- interfaces/IEthVaultFactory.sol
- interfaces/IERC20.sol
- interfaces/IVaultState.sol
- interfaces/IVaultValidators.sol
- interfaces/IMulticall.sol
- interfaces/IERC20Permit.sol
- interfaces/IKeeperRewards.sol
- interfaces/IEthPrivateVault.sol
- interfaces/IVaultAdmin.sol
- interfaces/IVaultFee.sol
- interfaces/ISharedMevEscrow.sol
- interfaces/IVaultImmutables.sol
- interfaces/IKeeperValidators.sol
- interfaces/IVaultEthStaking.sol
- interfaces/IEthValidatorsRegistry.sol
- interfaces/IEthVault.sol
- interfaces/IVaultToken.sol
- interfaces/IVaultsRegistry.sol
- interfaces/IKeeper.sol
- interfaces/IOwnMevEscrow.sol
- interfaces/IVaultMev.sol
- interfaces/IVaultEnterExit.sol
- interfaces/IVaultVersion.sol
- interfaces/IValidatorsRegistry.sol
- interfaces/IVersioned.sol

Out-of-scope:

- third-party libraries and dependencies
- economic attacks

2. Retest (May 4th, 2023)

- Repository: [v3-core](#)
- Commit ID: [7f6a034f2eec866771c5276773d97a5696331a3e](#)
- Branch: [vault-fixes](#)

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

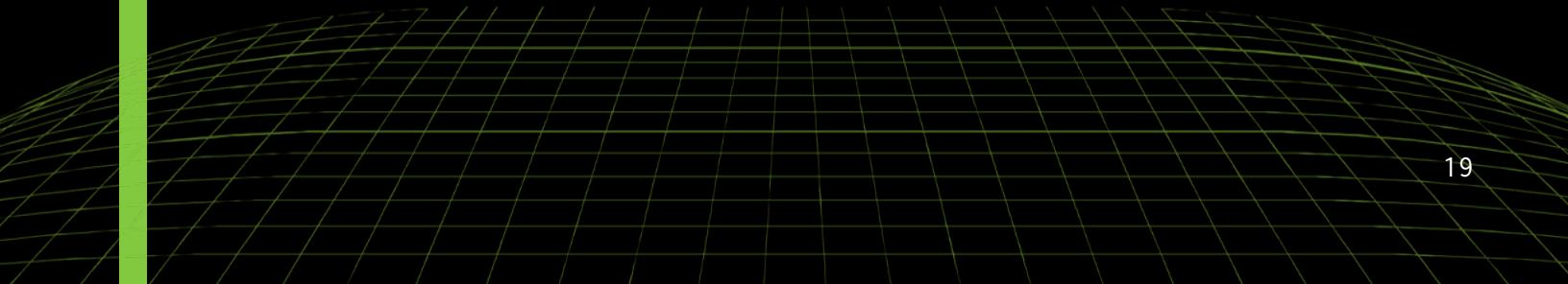
CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	3	1

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PRIVATE VAULT WHITELIST DEPOSIT BYPASS POSSIBLE	High (7.0)	SOLVED - 05/04/2023
LACK OF EMERGENCY-STOP PATTERN	Low (3.6)	RISK ACCEPTED
LACK OF ZERO ADDRESS CHECK FOR RECEIVER	Low (3.4)	SOLVED - 05/04/2023
VAULTADMIN LACKS OWNERSHIP-TRANSFER PATTERN	Low (3.0)	RISK ACCEPTED
INCREASEAPPROVE AND DECREASEAPPROVE CAN BE INTRODUCED	Informational (0.0)	SOLVED - 05/04/2023



FINDINGS & TECH DETAILS



4.1 (HAL-01) PRIVATE VAULT WHITELIST DEPOSIT BYPASS POSSIBLE - HIGH (7.0)

Description:

The solution allows deploying two types of vaults: `EthVault` and `EthPrivateVault`. The `EthPrivateVault` allows whitelisted participants to deposit only. However, the assessment revealed that a bypass of this limitation exists in the `EthPrivateVault` contract. Whereas the `deposit()` and the `updateStateAndDeposit()` functions are protected, the solution still allows making a deposit with the `receive()` function. The `receive()` function uses internal the `_deposit()` function instead of public `deposit()` which contains security constraints. Therefore, any account can deposit in the `EthPrivateVault`. The list of whitelisted participants is carefully created off-chain. Such a group of accounts may pose significant deposit that may ensure profit. This weakness makes the `EthPrivateVault` main functionality not relevant and may lead to profit share with any staker.

Code Location:

Listing 1: EthPrivateVault.sol

```

53   function deposit(
54     address receiver,
55     address referrer
56   ) public payable override(IVaultEthStaking, VaultEthStaking)
57     returns (uint256 shares) {
58     if (!(whitelistedAccounts[msg.sender] && whitelistedAccounts[
59       receiver])) revert AccessDenied();
60     return super.deposit(receiver, referrer);
61   }

```

Listing 2: VaultEthStaking.sol

```

39 function deposit(
40   address receiver,
41   address referrer

```

Proof of Concept:

1. Deploy the `EthPrivateVault` vault as an admin.
 2. As a non-whitelisted depositor, attempt to deposit any amount of ether by means of `deposit()` function.
 3. Observe that the above transaction reverts with the `AccessDenied` error.
 4. As a non-whitelisted depositor, attempt to deposit any amount of ether by sending the ether directly to the contract.
 5. Observe that the above transaction was completed successfully.

The below unit test implements the PoC:

```
Listing 3: VaultEthStaking.sol

39     function
↳ test_hal_EthPrivateVault_deposit_via_receive_as_non_whitelisted()
↳ public {
40     vm.startPrank(depositor1);
41
42     vm.expectRevert(IVaultAdmin.AccessDenied.selector);
43     ethPrivateVault.deposit{value: 1 ether}(depositor1,
↳ depositor1);
44
45     (bool success, ) = payable(address(ethPrivateVault)).call{
↳ value: 1 ether}());
46     if (!success) revert("Call failed.");
47     vm.stopPrank();
48 }
```

BVSS:

A0:A/AC:L/AX:L/C:C/I:N/A:N/D:N/Y:M/R:P/S:C (7.0)

Recommendation:

It is recommended to apply checks against whitelist within the `receive()` function for the `EthPrivateVault`.

Remediation Plan:

SOLVED: The StakeWise team solved this issue in commit `7f6a034f2eec866771c5276773d97a5696331a3e`: the `receive()` function in the `EthPrivateVault` is now protected with whitelist check.

4.2 (HAL-02) LACK OF EMERGENCY-STOP PATTERN - LOW (3.6)

Description:

The current solution does not implement any kind of `emergency stop` pattern. Such a pattern allows the project team to pause crucial functionalities, while being in the state of emergency, e.g. being under adversary attack. The most prevalent application of the `emergency stop` pattern is the the OpenZeppelin's `Pausable` library (or alternately `PausableUpgradeable`).

In this case, if the `emergency stop` pattern is not implemented, then functions such as `deposit()`, `withdraw()`, `setRewardsRoot()`, `enterExitQueue()` cannot be temporarily disabled.

BVSS:

A0:A/AC:L/AX:H/C:N/I:C/A:C/D:C/Y:C/R:P/S:C (3.6)

Recommendation:

It is recommended to implement `emergency stop` pattern across the solution.

Remediation Plan:

RISK ACCEPTED: The StakeWise team accepted the risk. Vaults will be used by various businesses. It is a business decision that vaults must be permissionless.

4.3 (HAL-03) LACK OF ZERO ADDRESS CHECK FOR RECEIVER - LOW (3.4)

Description:

The `EthVault` allows withdrawing deposits by the means of the `withdraw()`, `redeem()` and `enterExitQueue()` functions. All of these functions take two addresses as input parameters: `receiver` and `owner`. The list and order of parameters might be confusing and prone to human errors. E.g. end user may think that if no `receiver` is provided, the `owner` is set as receiver. Therefore, a transaction can be triggered that sends ether to the zero address.

In contrary, the `deposit()` function contains zero address check for `to` parameter.

Code Location:

Listing 4: VaultEnterExit.sol

```
22 /// @inheritdoc IVaultEnterExit
23   function withdraw(
24     uint256 assets,
25     address receiver,
26     address owner
27   ) external override returns (uint256 shares) {
28     // calculate amount of shares to burn
29     shares = _convertToShares(assets, _totalShares, _totalAssets,
30     ↳ Math.Rounding.Up);
31     _withdraw(receiver, owner, assets, shares);
32   }
33
34   /// @inheritdoc IVaultEnterExit
35   function redeem(
36     uint256 shares,
37     address receiver,
38     address owner
39   ) external override returns (uint256 assets) {
40     // calculate amount of assets to burn
41     assets = convertToAssets(shares);
```

```
41     _withdraw(receiver, owner, assets, shares);
42 }
43
44 /// @inheritdoc IVaultEnterExit
45 function enterExitQueue(
46     uint256 shares,
47     address receiver,
48     address owner
49 ) external override returns (uint256 positionCounter) {
50     if (shares == 0) revert InvalidSharesAmount();
51     if (!IKeeperRewards(keeper).isCollateralized(address(this)))
52         revert NotCollateralized();
53     // SLOAD to memory
54     uint256 _queuedShares = queuedShares;
55
56     // calculate position counter
57     positionCounter = _exitQueue.getSharesCounter() +
58         _queuedShares;
59
60     // add to the exit requests
61     _exitRequests[keccak256(abi.encode(receiver, positionCounter))]
62     ] = shares;
63
64     // lock tokens in the Vault
65     if (msg.sender != owner) _spendAllowance(owner, msg.sender,
66         shares);
67     // reverts if owner does not have enough shares
68     balanceOf[owner] -= shares;
69
70     unchecked {
71         // cannot overflow as it is capped with _totalShares
72         queuedShares = SafeCast.toInt96(_queuedShares + shares);
73     }
74
75     emit Transfer(owner, address(this), shares);
76     emit ExitQueueEntered(msg.sender, receiver, owner,
77         positionCounter, shares);
78 }
```

Listing 5: VaultEnterExit.sol

```
166 function _withdraw(address receiver, address owner, uint256 assets
167 , uint256 shares) internal {
```

```
167      if (IKeeperRewards(keeper).isHarvestRequired(address(this)))
168        ↳ revert NotHarvested();
169
170      // reverts in case there are not enough withdrawable assets
171      if (assets > withdrawableAssets()) revert InsufficientAssets()
172        ↳ ;
173
174      // reduce allowance
175      if (msg.sender != owner) _spendAllowance(owner, msg.sender,
176        ↳ shares);
177
178      // burn shares
179      balanceOf[owner] -= shares;
180
181      // update counters
182      unchecked {
183        // cannot underflow because the sum of all shares can't
184        ↳ exceed the _totalShares
185        _totalShares -= SafeCast.toInt128(shares);
186        // cannot underflow because the sum of all assets can't
187        ↳ exceed the _totalAssets
188        _totalAssets -= SafeCast.toInt128(assets);
189      }
190
191      // transfer assets to the receiver
192      _transferVaultAssets(receiver, assets);
193
194      emit Transfer(owner, address(0), shares);
195      emit Withdraw(msg.sender, receiver, owner, assets, shares);
196    }
```

BVSS:

A0:A/AC:L/AX:M/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (3.4)

Recommendation:

It is recommended to add a zero address check against the `receiver` parameter in the `withdraw()`, `redeem()` and `enterExitQueue()` functions.

FINDINGS & TECH DETAILS

Remediation Plan:

SOLVED: The StakeWise team solved this issue in commit [665711457df10c8a4f1e9ff8d3881a52175757c6](#): the zero address check against the `receiver` parameter is now added to the `redeem()` and `enterExitQueue()` functions. The `withdraw()` function was removed.

4.4 (HAL-04) VAULTADMIN LACKS OWNERSHIP-TRANSFER PATTERN - LOW (3.0)

Description:

The `EthVault` allows setting an `admin` account during the vault's deployment. However, the assessment revealed that there is no possibility to update the `admin` account within the vault. In the case of account highjacking, multiple functionalities get under permanent control of the attacker, including `setFeeRecipient()`, `setOperator()`, `upgradeToAndCall()`, `setWhitelister()`, `setMetadata()`.

In contrary, the `VaultsRegistry` and `Keeper` solutions implement a two-steps ownership-transfer pattern.

BVSS:

A0: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 StakeWise team accepted the risk. It is a business decision that vault admin privileges are not transferable. In the future, vault admins can verify themselves through social media (e.g., posting the signature of a signed message on Twitter). Users can be affected if they stake to the vault created by one entity and later transferred to another.

4.5 (HAL-05) INCREASEAPPROVE AND DECREASEAPPROVE CAN BE INTRODUCED - INFORMATIONAL (0.0)

Description:

The `EthVault` implements the `ERC20Upgradeable` standard. To manage allowance, the solution implements both `approve()` and `permit()` functions. However, it does not implement non-standard `increaseAllowance()` and `decreaseAllowance()` functions. These functions mitigate the possible front-running of `approve()` function, where someone may use both the old and the new allowance.

BVSS:

A0:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

It is recommended to implement non-standard `increaseAllowance()` and `decreaseAllowance()` functions within the `ERC20Upgradeable` contract.

Remediation Plan:

SOLVED: The StakeWise team solved this issue in commit [7f6a034f2eec866771c5276773d97a5696331a3e](#): the `ERC20Upgradeable` now implements non-standard `increaseAllowance()` and `decreaseAllowance()` functions.

MANUAL TESTING

In the manual testing phase, multiple unit tests were written in the **Foundry** framework. Various scenarios were simulated depending on the context and tested smart contract.

The purpose of **OwnMevEscrow** and **SharedMevEscrow** unit testing was to verify if contracts business logic works as expected along with the implemented authorization.

```
Running 11 tests for test/HalbornEscrowTest.t.sol:HalbornEscrowTest
[PASS] test_hal_OwnMevEscrow_harvest_as_non_vault_reverts() (gas: 147613)
[PASS] test_hal_OwnMevEscrow_harvest_balance() (gas: 329642)
[PASS] test_hal_OwnMevEscrow_harvest_balance_as_non_IVaultEthStaking_reverts() (gas: 155764)
[PASS] test_hal_OwnMevEscrow_harvest_zero() (gas: 145240)
[PASS] test_hal_OwnMevEscrow_transfer_ether_to_mev() (gas: 154318)
[PASS] test_hal_SharedMevEscrow_harvest_above_balance_reverts() (gas: 340482)
[PASS] test_hal_SharedMevEscrow_harvest_as_non_vault_reverts() (gas: 157193)
[PASS] test_hal_SharedMevEscrow_harvest_balance() (gas: 340254)
[PASS] test_hal_SharedMevEscrow_harvest_half_of_balance() (gas: 340254)
[PASS] test_hal_SharedMevEscrow_harvest_zero() (gas: 324174)
[PASS] test_hal_SharedMevEscrow_transfer_ether_to_mev() (gas: 163899)
Test result: ok. 11 passed; 0 failed; finished in 6.71ms
```

The purpose of **EthPrivateVault** unit testing was to verify if contract business logic works as expected along with the implemented authorization. An attempt to bypass the white-list was implemented. Additionally, a scenario was identified where a user can call **joinWhitelist()** multiple times to add himself/herself to the list after delisting. However, the StakeWise team confirmed that it is not considered as an issue.

```
Running 20 tests for test/HalbornEthPrivateVaultTest.t.sol:HalbornEthPrivateVaultTest
[PASS] test_hal_EthPrivateVault_attempt_to_initilize_twice_reverts() (gas: 26313)
[PASS] test_hal_EthPrivateVault_deposit_as_non_whitelisted() (gas: 143277)
[PASS] test_hal_EthPrivateVault_deposit_as_owner_valid_scenario() (gas: 188827)
[PASS] test_hal_EthPrivateVault_deposit_as_owner_with_empty_receiver_reverts() (gas: 145570)
[PASS] test_hal_EthPrivateVault_deposit_as_owner_with_empty_referrer_valid_scenario() (gas: 188802)
[PASS] test_hal_EthPrivateVault_deposit_valid_scenario() (gas: 71588)
[PASS] test_hal_EthPrivateVault_deposit_via_receive_as_non_whitelisted_valid_scenario() (gas: 185644)
[PASS] test_hal_EthPrivateVault_deposit_via_receive_as_owner_valid_scenario() (gas: 185635)
[PASS] test_hal_EthPrivateVault_joinWhitelist_for_multiple_addresses_valid_scenario() (gas: 685984)
[PASS] test_hal_EthPrivateVault_joinWhitelist_for_one_address_then_updateWhitelist_with_false_then_JoinWhitelist() (gas: 90184)
[PASS] test_hal_EthPrivateVault_joinWhitelist_for_one_address_valid_scenario() (gas: 76107)
[PASS] test_hal_EthPrivateVault_joinWhitelist_twice_in_a_row_reverts() (gas: 587690)
[PASS] test_hal_EthPrivateVault_setWhitelistRoot_as_non_admin() (gas: 90959)
[PASS] test_hal_EthPrivateVault_setWhitelister_as_non_admin() (gas: 58574)
[PASS] test_hal_EthPrivateVault_setWhitelister_twice_valid_scenario() (gas: 72599)
[PASS] test_hal_EthPrivateVault_setWhitelister_valid_scenario() (gas: 68350)
[PASS] test_hal_EthPrivateVault_updateWhitelist_as_non_admin() (gas: 90894)
[PASS] test_hal_EthPrivateVault_updateWhitelist_non_existing_to_false_reverts() (gas: 29686)
[PASS] test_hal_EthPrivateVault_updateWhitelist_twice_reverts() (gas: 55313)
[PASS] test_hal_EthPrivateVault_updateWhitelist_valid_scenario() (gas: 42708)
Test result: ok. 20 passed; 0 failed; finished in 9.36ms
```

The purpose of `EthVaultFactory` unit testing was to verify if contract business logic works as expected.

```
Running 7 tests for test/HalbornEthVaultFactoryTest.t.sol:HalbornEthVaultFactoryTest
[PASS] test_hal_EthVaultFactory_compare_computeAddresses_for_the_same_deployer() (gas: 17946)
[PASS] test_hal_EthVaultFactory_compare_computeAddresses_for_two_deployers() (gas: 21925)
[PASS] test_hal_EthVaultFactory_compare_computeAddresses_for_two_deployers_and_private_vault() (gas: 22009)
[PASS] test_hal_EthVaultFactory_deploy_and_compare_computeAddresses() (gas: 459649)
[PASS] test_hal_EthVaultFactory_deploy_too_high_feePrecent_reverts() (gas: 272206)
[PASS] test_hal_EthVaultFactory_deploy_too_low_security_deposit_reverts() (gas: 355624)
[PASS] test_hal_EthVaultFactory_deploy_too_small_capacity_reverts() (gas: 307104)
Test result: ok. 7 passed; 0 failed; finished in 10.33ms
```

The purpose of `MultiCall` unit testing was to verify if contract business logic works as expected.

```
Running 3 tests for test/HalbornMultiCallTest.t.sol:HalbornMultiCallTest
[PASS] test_hal_multicall_poc_1_approve_transfer() (gas: 338250)
[PASS] test_hal_multicall_poc_2_empty_deposit_possible() (gas: 257876)
[PASS] test_hal_multicall_poc_3_redeem_twice() (gas: 448591)
Test result: ok. 3 passed; 0 failed; finished in 10.88ms
```

The purpose of `EthVault` unit testing was to verify if contract business logic works as expected along with the implemented authorization. This part of the unit test included multiple various functionalities from vault's modules, that were excluded in other tests. The upgradability functionality was also included.

```
Running 24 tests for test/HalbornEthVaultTest.t.sol:HalbornEthVaultTest
[PASS] test_hal_EthVault_attempt_to_initilize_twice_reverts() (gas: 26290)
[PASS] test_hal_EthVault_receiveFromMevEscrow_as_non_mev_reverts() (gas: 25720)
[PASS] test_hal_EthVault_setFeeRecipient_as_non_owner_reverts() (gas: 18535)
[PASS] test_hal_EthVault_setFeeRecipient_as_zero_address_reverts() (gas: 29150)
[PASS] test_hal_EthVault_setFeeRecipient_valid_scenario() (gas: 37328)
[PASS] test_hal_EthVault_setMetadata_as_admin() (gas: 20402)
[PASS] test_hal_EthVault_setMetadata_as_non_admin_reverts() (gas: 18773)
[PASS] test_hal_EthVault_setOperator_as_not_an_owner_reverts() (gas: 18514)
[PASS] test_hal_EthVault_setOperator_as_owner_valid_scenario() (gas: 43876)
[PASS] test_hal_EthVault_setOperator_as_zero_address_reverts() (gas: 18562)
[PASS] test_hal_EthVault_setOperator_then_setOperator_back_to_owner() (gas: 52326)
[PASS] test_hal_EthVault_setOperator_then_setOperator_to_zero_address_reverts() (gas: 47596)
[PASS] test_hal_EthVault_setValidatorsRoot_as_not_an_owner_reverts() (gas: 20643)
[PASS] test_hal_EthVault_setValidatorsRoot_twice_valid_scenario() (gas: 27446)
[PASS] test_hal_EthVault_setValidatorsRoot_valid_scenario() (gas: 24141)
[PASS] test_hal_EthVault_upgradeToAndCall_incorrect_vaultId_reverts() (gas: 95211)
[PASS] test_hal_EthVault_upgradeToAndCall_initialize_twice_reverts() (gas: 4191791)
[PASS] test_hal_EthVault_upgradeToAndCall_no_implementation_added_to_registry_reverts() (gas: 4103899)
[PASS] test_hal_EthVault_upgradeToAndCall_the_same_implementation_reverts() (gas: 42126)
[PASS] test_hal_EthVault_upgradeToAndCall_valid_scenario() (gas: 4206520)
[PASS] test_hal_EthVault_upgradeToAndCall_zero_address_not_as_admin_reverts() (gas: 36189)
[PASS] test_hal_EthVault_upgradeToAndCall_zero_address_reverts() (gas: 36219)
[PASS] test_hal_EthVault_upgradeTo_zero_address_not_as_admin_reverts() (gas: 33518)
[PASS] test_hal_EthVault_upgradeTo_zero_address_reverts() (gas: 33502)
Test result: ok. 24 passed; 0 failed; finished in 11.42ms
```

The purpose of `Oracles` unit testing was to verify if contracts business logic works as expected along with the implemented authorization. Multiple use cases were written to verify if a list of oracles can be properly maintained.

```
Running 14 tests for test/HalbornOraclesTest.t.sol:HalbornOraclesTest
[PASS] test_hal_Oracles_addOracle_as_not_owner() (gas: 11613)
[PASS] test_hal_Oracles_addOracle_as_owner() (gas: 43557)
[PASS] test_hal_Oracles_addOracle_same_address_twice_as_owner() (gas: 45817)
[PASS] test_hal_Oracles_addOracle_then_removeOracle_as_owner() (gas: 35781)
[PASS] test_hal_Oracles_removeOracle_as_not_owner() (gas: 11622)
[PASS] test_hal_Oracles_removeOracle_for_all_oracles_as_owner() (gas: 112855)
[PASS] test_hal_Oracles_removeOracle_for_more_than_requiredOracles_as_owner() (gas: 77949)
[PASS] test_hal_Oracles_removeOracle_for_ten_oracles_as_owner() (gas: 106137)
[PASS] test_hal_Oracles_removeOracle_non_existing_as_owner() (gas: 19659)
[PASS] test_hal_Oracles_setRequiredOracles_as_not_owner() (gas: 13867)
[PASS] test_hal_Oracles_setRequiredOracles_as_owner() (gas: 22599)
[PASS] test_hal_Oracles_setRequiredOracles_as_zero_owner() (gas: 16052)
[PASS] test_hal_Oracles_setRequiredOracles_for_more_than_totalOracles_as_owner() (gas: 19080)
[PASS] test_hal_Oracles_setRequiredOracles_for_totalOracles_as_owner() (gas: 24122)
Test result: ok. 14 passed; 0 failed; finished in 7.95ms
```

In the following unit testing exercise, multiple scenarios were written to verify if deposit and withdraw/redeem functionalities work as expected. The accounting of shares and assets were checked manually.

```
Running 15 tests for test/HalbornDepositTest.t.sol:HalbornDepositTest
[PASS] test_hal_deposit() (gas: 269662)
[PASS] test_hal_deposit_max() (gas: 283331)
[PASS] test_hal_deposit_multiple_deposits() (gas: 338289)
[PASS] test_hal_deposit_one() (gas: 269708)
[PASS] test_hal_deposit_receiver_is_zero_address() (gas: 246263)
[PASS] test_hal_deposit_referrer_as_depositor() (gas: 269664)
[PASS] test_hal_deposit_then_redeem() (gas: 419564)
[PASS] test_hal_deposit_then_redeem_balance_minus_one() (gas: 443786)
[PASS] test_hal_deposit_then_redeem_one() (gas: 442288)
[PASS] test_hal_deposit_then_redeem_to_zero_address() (gas: 447059)
[PASS] test_hal_deposit_then_withdraw() (gas: 419550)
[PASS] test_hal_deposit_then_withdraw_balance_minus_one() (gas: 443805)
[PASS] test_hal_deposit_then_withdraw_one() (gas: 442249)
[PASS] test_hal_deposit_then_withdraw_to_zero_address() (gas: 447020)
[PASS] test_hal_deposit_zero_deposit_is_possible() (gas: 239946)
Test result: ok. 15 passed; 0 failed; finished in 23.13ms
```

The purpose of `VaultsRegistry` unit testing was to verify if contract business logic works as expected along with the implemented authorization.

```
Running 15 tests for test/HalbornVaultsRegistryTest.t.sol:HalbornVaultsRegistryTest
[PASS] test_hal_addFactory_as_not_an_owner_reverts() (gas: 13564)
[PASS] test_hal_addFactory_as_owner() (gas: 36219)
[PASS] test_hal_addFactory_as_owner_twice_reverts() (gas: 37902)
[PASS] test_hal_addFactory_then_addVault() (gas: 63853)
[PASS] test_hal_addFactory_then_addVault_twice() (gas: 66638)
[PASS] test_hal_addFactory_then_removeFactory_as_owner() (gas: 26876)
[PASS] test_hal_addFactory_then_removeFactory_twice_as_owner_reverts() (gas: 28248)
[PASS] test_hal_addVaultImpl_as_not_an_owner_reverts() (gas: 13519)
[PASS] test_hal_addVaultImpl_as_owner() (gas: 36274)
[PASS] test_hal_addVaultImpl_as_owner_twice_reverts() (gas: 37899)
[PASS] test_hal_addVaultImpl_then_removeVaultImpl_as_owner() (gas: 26895)
[PASS] test_hal_addVaultImpl_then_removeVaultImpl_twice_as_owner_reverts() (gas: 28168)
[PASS] test_hal_addVault_reverts() (gas: 13143)
[PASS] test_hal_removeFactory_as_not_an_owner_reverts() (gas: 13578)
[PASS] test_hal_removeVaultImpl_as_not_an_owner_reverts() (gas: 13565)
Test result: ok. 15 passed; 0 failed; finished in 24.13ms
```

The purpose of `Keeper` unit testing was to verify if contract business logic works as expected along with the implemented authorization. The upgradability functionality was also included. Various scenarios related to signature handling were also included.

```
Running 15 tests for test/HalbornKeeperTest.t.sol:HalbornKeeperTest
[PASS] test_hal_KeeperRewards_attempt_to_initilize_twice_reverts() (gas: 19706)
[PASS] test_hal_KeeperRewards_initialize_twice_reverts() (gas: 19127)
[PASS] test_hal_KeeperRewards_setRewardsDelay_as_non_owner_reverts() (gas: 18584)
[PASS] test_hal_KeeperRewards_setRewardsDelay_as_owner_to_one_day() (gas: 30636)
[PASS] test_hal_KeeperRewards_setRewardsDelay_as_owner_to_zero() (gas: 30646)
[PASS] test_hal_Keeper_upgradeToAndCall_v2mock_valid_scenario() (gas: 2199983)
[PASS] test_hal_Keeper_upgradeToAndCall_zero_address_not_as_admin_reverts() (gas: 27182)
[PASS] test_hal_Keeper_upgradeToAndCall_zero_address_reverts() (gas: 31980)
[PASS] test_hal_Keeper_upgradeTo_v2mock_valid_scenario() (gas: 2177392)
[PASS] test_hal_Keeper_upgradeTo_zero_address_not_as_admin_reverts() (gas: 26782)
[PASS] test_hal_Keeper_upgradeTo_zero_address_reverts() (gas: 31705)
[PASS] test_hal_updateExitSignatures_twice_reverts() (gas: 662011)
[PASS] test_hal_updateExitSignatures_twice_with_new_nonce() (gas: 745289)
[PASS] test_hal_updateExitSignatures_valid_scenario() (gas: 647610)
[PASS] test_hal_updateExitSignatures_without_validator_registered_reverts() (gas: 326048)
Test result: ok. 15 passed; 0 failed; finished in 72.36s
```

In the fallowing unit testing exercise, multiple scenarios were written to verify if:

- Oracles signature verification works as expected,
- validator registration works as expected,
- multiple validator registration works as expected,
- set rewards root works as expected,
- state update, harvest and state update and deposit work as expected,
- the accounting related to `ExitQueue` works as expected,
- the accounting differences between standard withdrawal and claim assets

from `ExitQueue` exists,

- the possibility to overwrite the `ExitQueue` records exists,
- the accounting in case of profit and loss works as expected.

Additionally, a scenario was identified where `harvest` reverts due to integer underflow. Whenever a subsequent call to `harvest` has set a smaller mev reward, this issue occurs. However, the StakeWise team confirmed that it is not considered as an issue as `HarvestParams` are strictly controlled off-chain and subsequent call cannot have smaller mev reward compared to previous.

```
Running 44 tests for test/HalbornValidatorTest.t.sol:HalbornValidatorTest
[PASS] test_hal_ValidatorsRegistry_get_deposit_root() (gas: 105281)
[PASS] test_hal_oracles_verifyAllSignatures_all_duplicated_signatures_reverts() (gas: 244630)
[PASS] test_hal_oracles_verifyAllSignatures_empty_exitSignaturesIpfHash_reverts() (gas: 235566)
[PASS] test_hal_oracles_verifyAllSignatures_incorrect_depositRoot_reverts() (gas: 234918)
[PASS] test_hal_oracles_verifyAllSignatures_incorrect_validators_data_revert() (gas: 234300)
[PASS] test_hal_oracles_verifyAllSignatures_incorrect_vault_address_reverts() (gas: 233915)
[PASS] test_hal_oracles_verifyAllSignatures_last_signature_duplicated_reverts() (gas: 297331)
[PASS] test_hal_oracles_verifyAllSignatures_only_one_signature_reverts() (gas: 225577)
[PASS] test_hal_oracles_verifyAllSignatures_signatures_incorrect_order_reverts() (gas: 272910)
[PASS] test_hal_oracles_verifyAllSignatures_signatures_not_long_enough_reverts() (gas: 229154)
[PASS] test_hal_oracles_verifyAllSignatures_six_signatures_reverts() (gas: 226769)
[PASS] test_hal_oracles_verifyAllSignatures_valid_scenario() (gas: 294373)
[PASS] test_hal_registerValidator_attempt_to_register_twice_reverts() (gas: 972052)
[PASS] test_hal_registerValidator_invalid_proof_reverts() (gas: 842338)
[PASS] test_hal_registerValidator_not_enough_deposit_reverts() (gas: 837138)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_claimExitedAssets_valid_scenario() (gas: 1763288)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_100_ether_then_claimExitedAssets_valid_scenario() (gas: 2640502)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_attempt_to_override_queue() (gas: 2908970)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_but_reverse_order_valid_scenario() (gas: 2640764)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_then_deposit_valid_scenario() (gas: 2856781)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_valid_scenario() (gas: 2631871)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_withdraw_versus_claimExited() (gas: 3069577)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_then_claimExitedAssets_withdraw_versus_claimExited_but_withdraw_withdrawableAssets_before() (gas: 2901874)
[PASS] test_hal_registerValidator_then_enterExitQueue_then_updateState_with_loss_then_claimExitedAssets_valid_scenario() (gas: 2607630)
[PASS] test_hal_registerValidator_then_enterExitQueue_valid_scenario() (gas: 1377319)
[PASS] test_hal_registerValidator_then_setRewardsRoot_quadrice_various_but_fourth_as_first() (gas: 2279489)
[PASS] test_hal_registerValidator_then_setRewardsRoot_then_deposit() (gas: 1494989)
[PASS] test_hal_registerValidator_then_setRewardsRoot_then_harvest_then_enterExitQueue_valid_scenario() (gas: 1181476)
[FAIL, Reason: Arithmetic over/underflow] test_hal_registerValidator_then_setRewardsRoot_then_harvest_then_setRewardsRoot_then_harvest_integ
er_underflow_reverts() (gas: 1888821)
[PASS] test_hal_registerValidator_then_setRewardsRoot_then_harvest_twice() (gas: 1722302)
[PASS] test_hal_registerValidator_then_setRewardsRoot_then_harvest_valid_scenario() (gas: 1718481)
[PASS] test_hal_registerValidator_then_setRewardsRoot_then_updateStateAndDeposit_valid_scenario() (gas: 1753828)
[PASS] test_hal_registerValidator_then_setRewardsRoot_twice_the_same_rewardFreeRoot_reverts() (gas: 1323466)
[PASS] test_hal_registerValidator_then_setRewardsRoot_twice_various() (gas: 1437551)
[PASS] test_hal_registerValidator_then_setRewardsRoot_twice_various_then_deposit_reverts() (gas: 1815005)
[PASS] test_hal_registerValidator_then_setRewardsRoot_valid_scenario() (gas: 1276039)
[PASS] test_hal_registerValidator_then_updateState_with_loss_then_withdraw_valid_scenario() (gas: 2125755)
[PASS] test_hal_registerValidator_valid_scenario() (gas: 927762)
[PASS] test_hal_registerValidator_valid_scenario_then_setRewardsRoot_too_early_reverts() (gas: 1219506)
[PASS] test_hal_registerValidators_ten_incorrect_indexes_revert() (gas: 1209609)
[PASS] test_hal_registerValidators_ten_incorrect_proofFlags_revert() (gas: 1202423)
[PASS] test_hal_registerValidators_ten_incorrect_validators_revert() (gas: 1197345)
[PASS] test_hal_registerValidators_ten_not_enough_deposit_reverts() (gas: 991955)
[PASS] test_hal_registerValidators_ten_valid_scenario() (gas: 1406599)
Test result: FAILED; 43 passed; 1 failed; finished in 88.08s
```



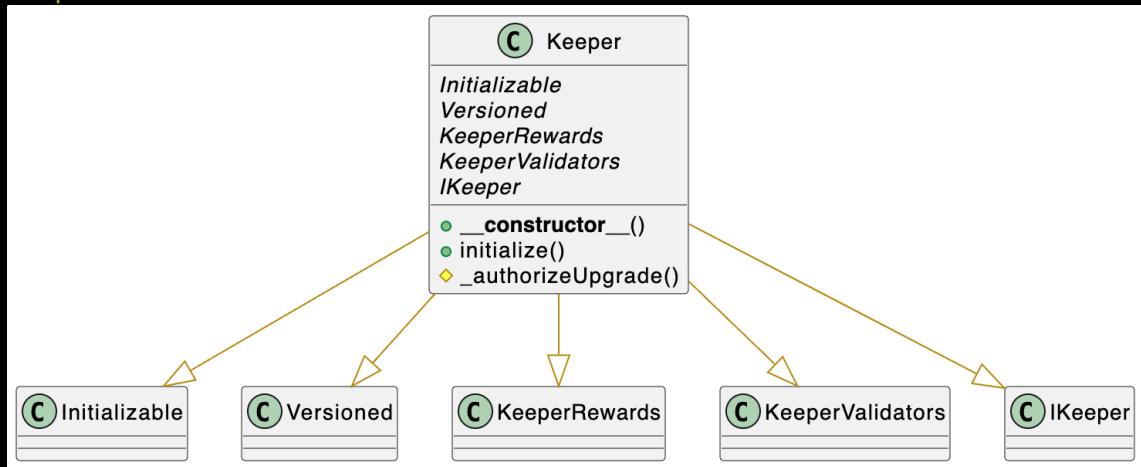
CONTRACT UPGRADABILITY



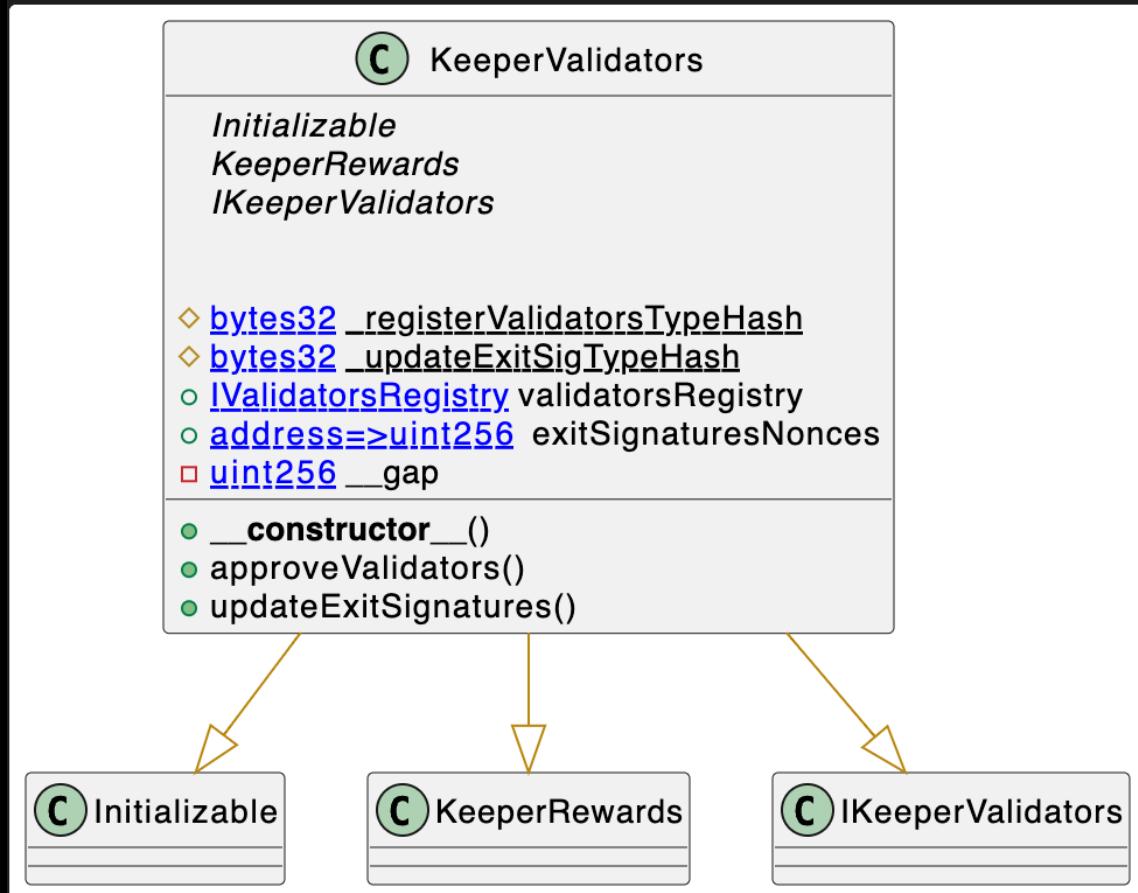
6.1 Solution Structure

The team at Halborn analyzed the structure of the smart contracts in scope to make sure all future upgrades are secure. The StakeWise team decided to use the `ERC1967Proxy` and `UUPSUpgradeable` within the solution. The `EthVault`, `EthPrivateVault` and `Keeper` contracts implement the `UUPSUpgradeable`. The `EthVaultFactory` contract uses `ERC1967Proxy` for deploying the vaults' implementations.

`Keeper.sol`



CONTRACT UPGRADABILITY

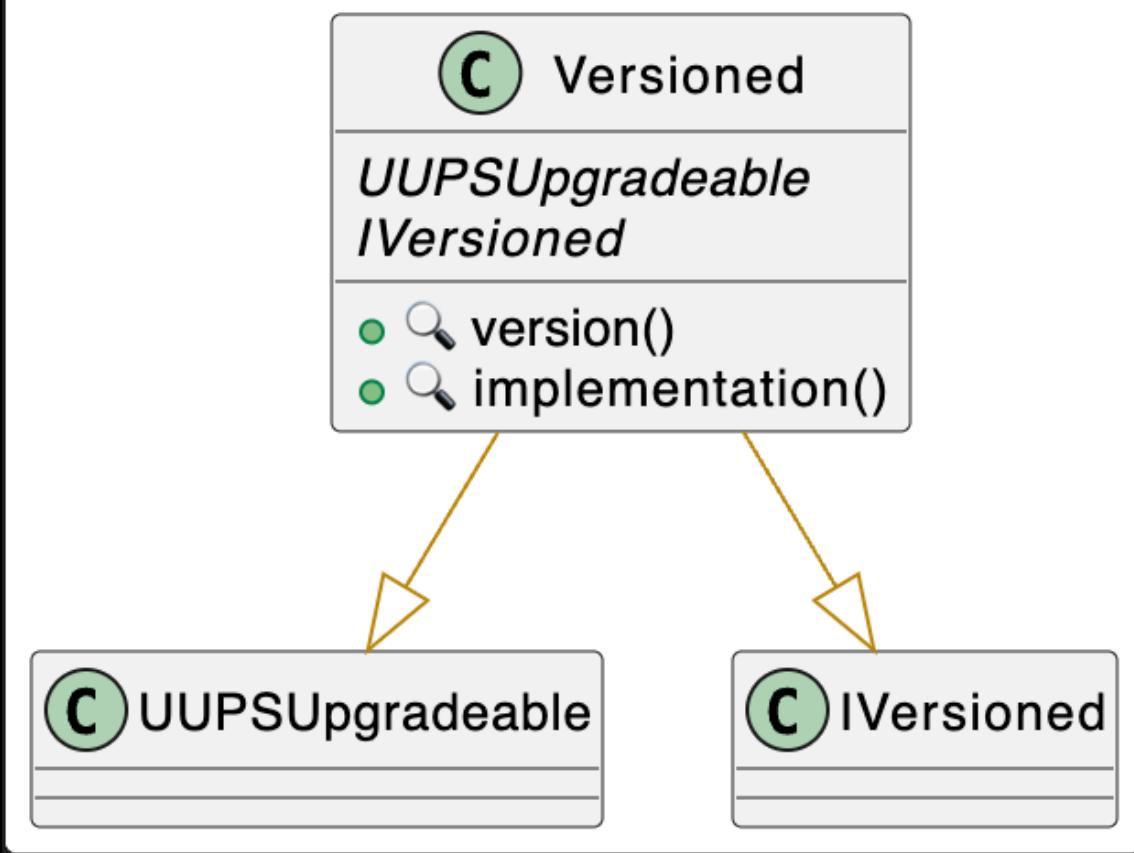
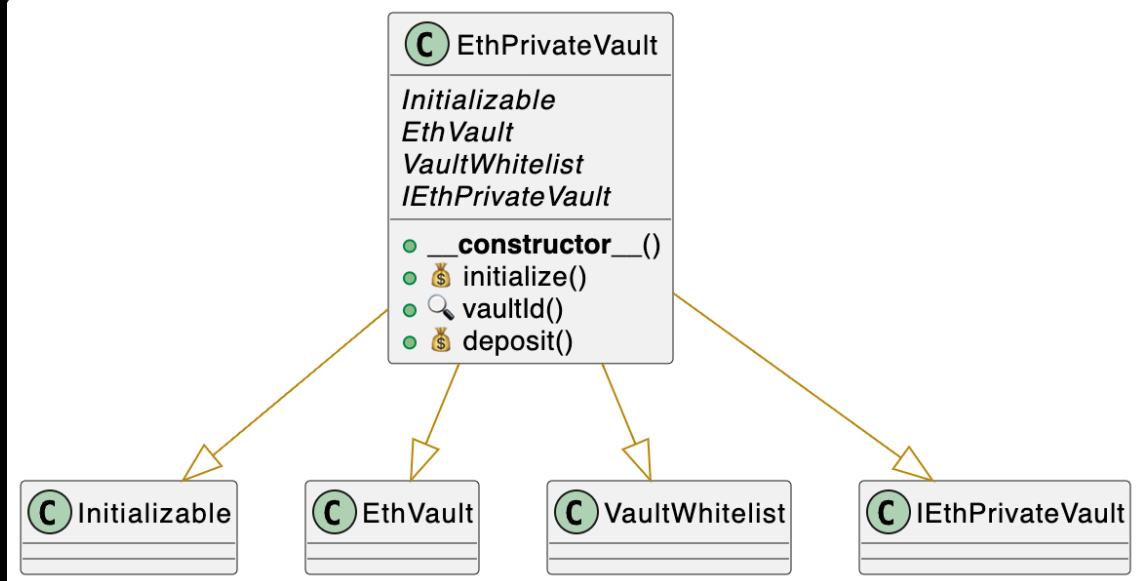


CONTRACT UPGRADABILITY

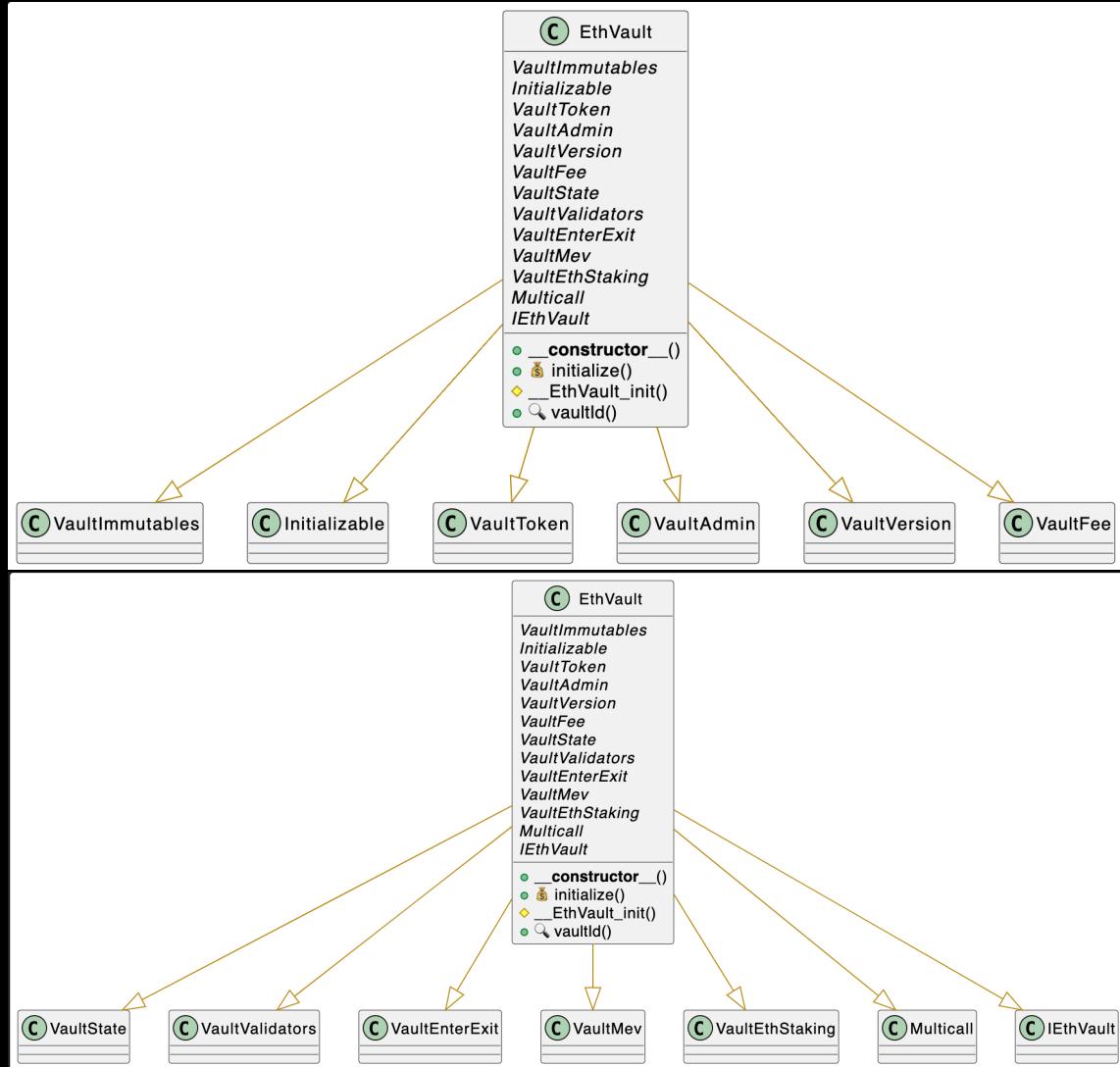


CONTRACT UPGRADABILITY

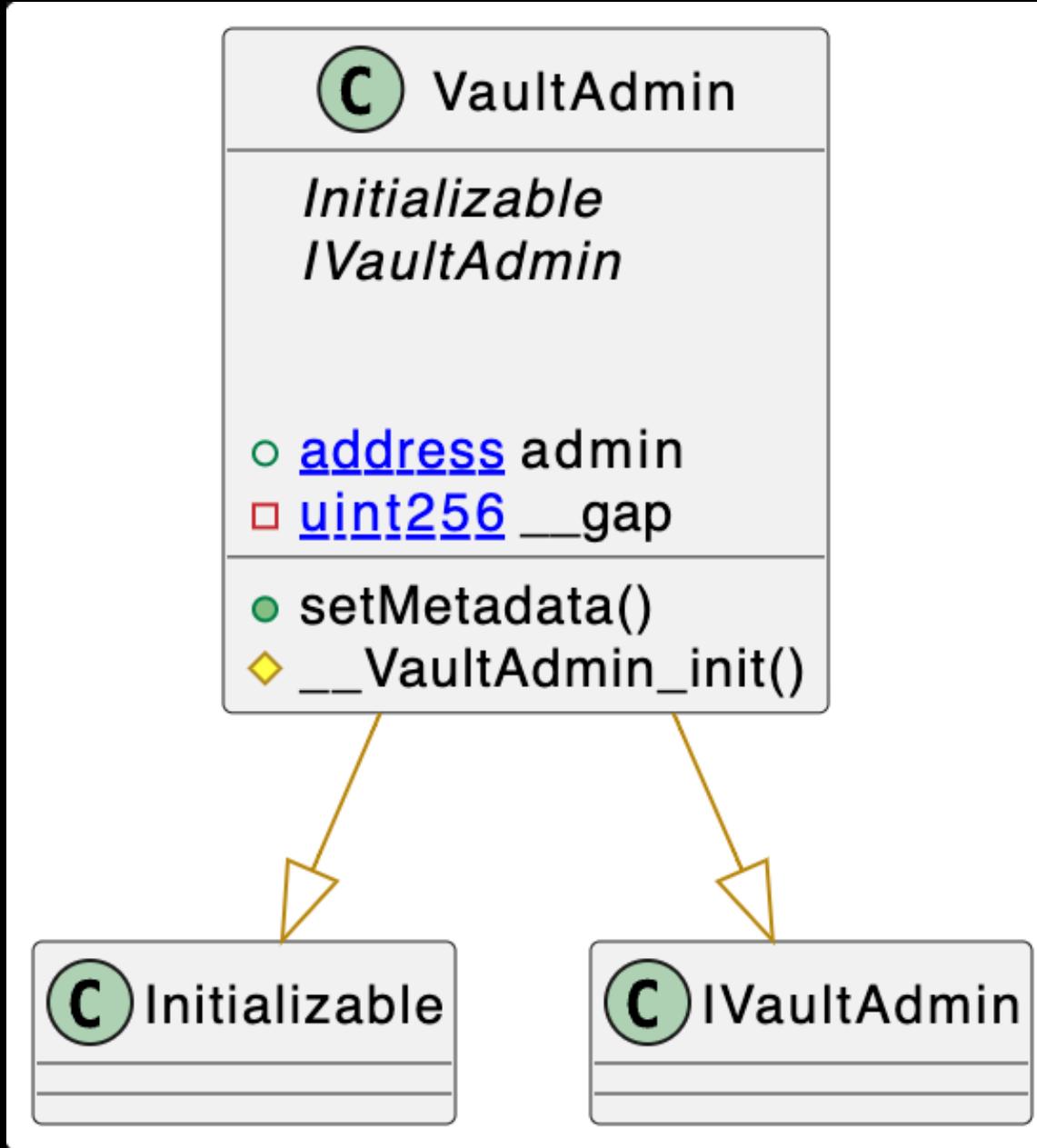
EthPrivateVault.sol



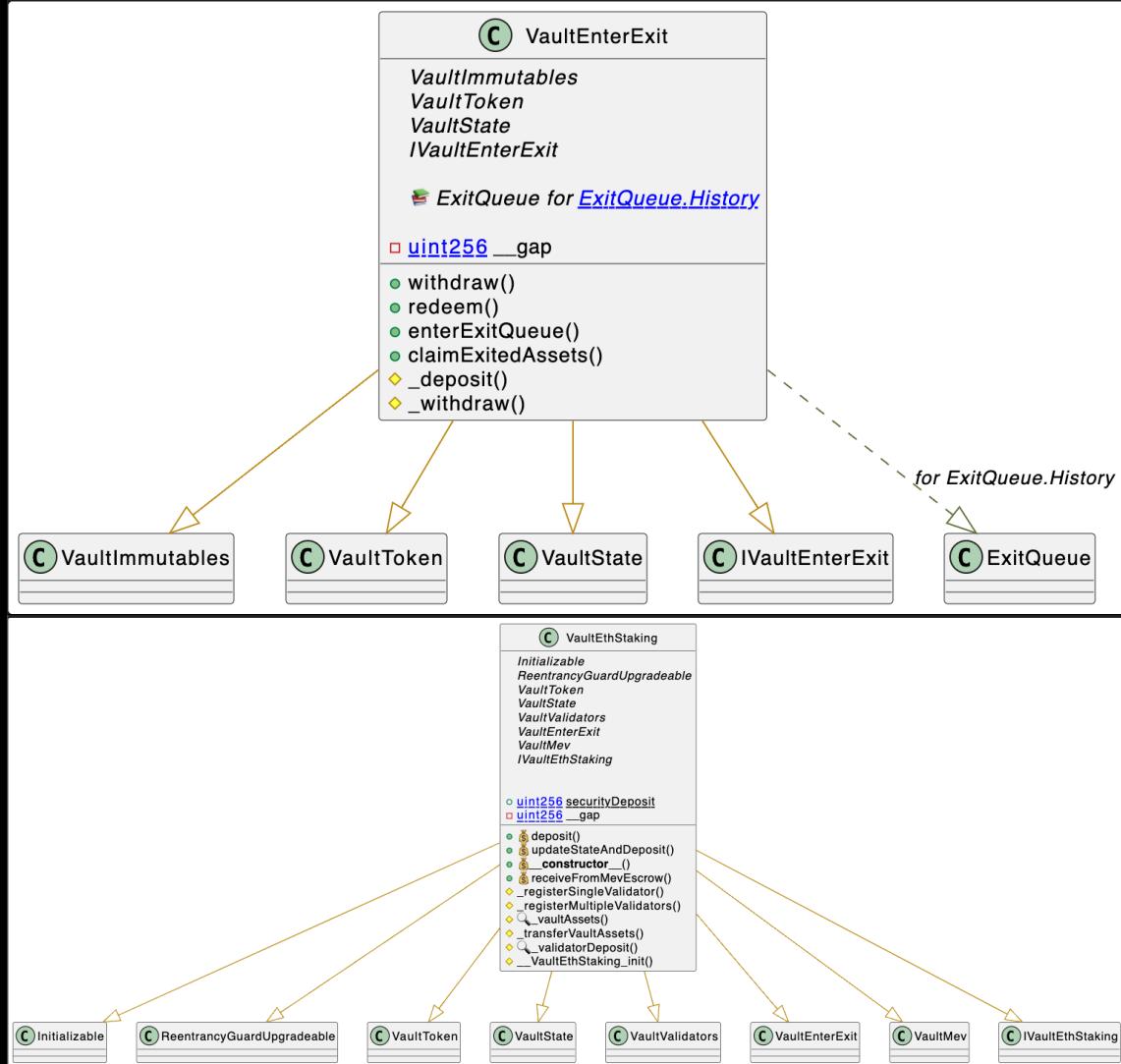
CONTRACT UPGRADABILITY



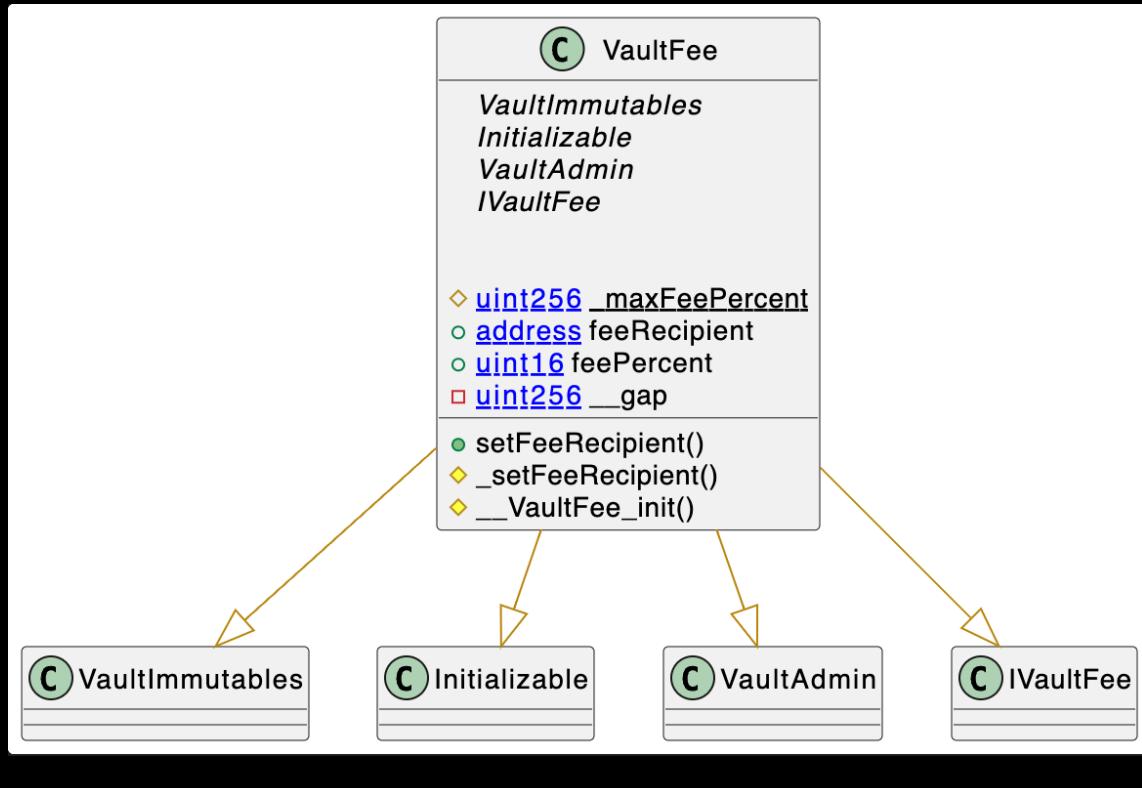
CONTRACT UPGRADABILITY



CONTRACT UPGRADABILITY



CONTRACT UPGRADABILITY





VaultImmutables

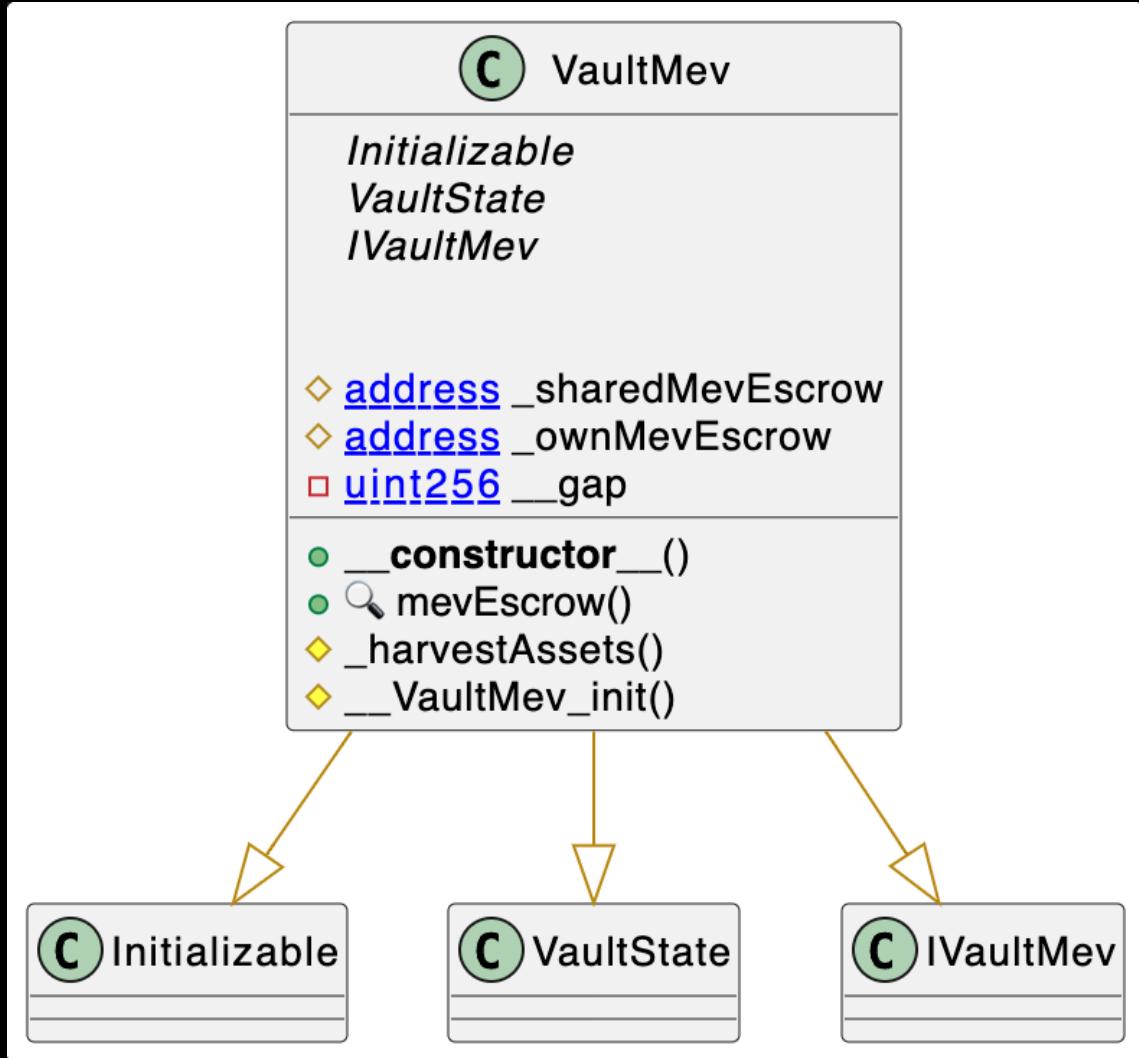
IVaultImmutables

- [address](#) keeper
- [address](#) vaultsRegistry
- [address](#) validatorsRegistry
- **constructor** ()

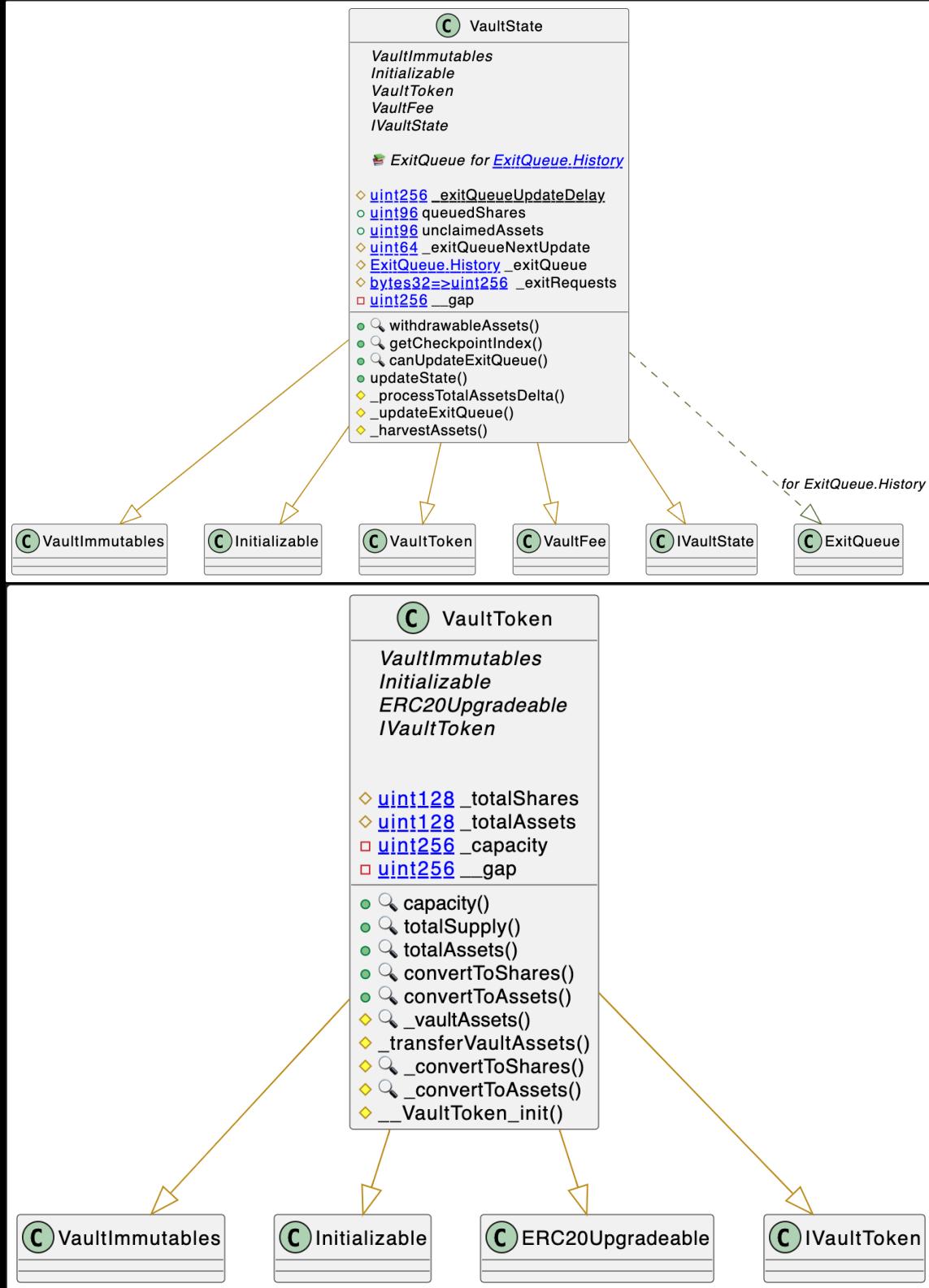


IVaultImmutables

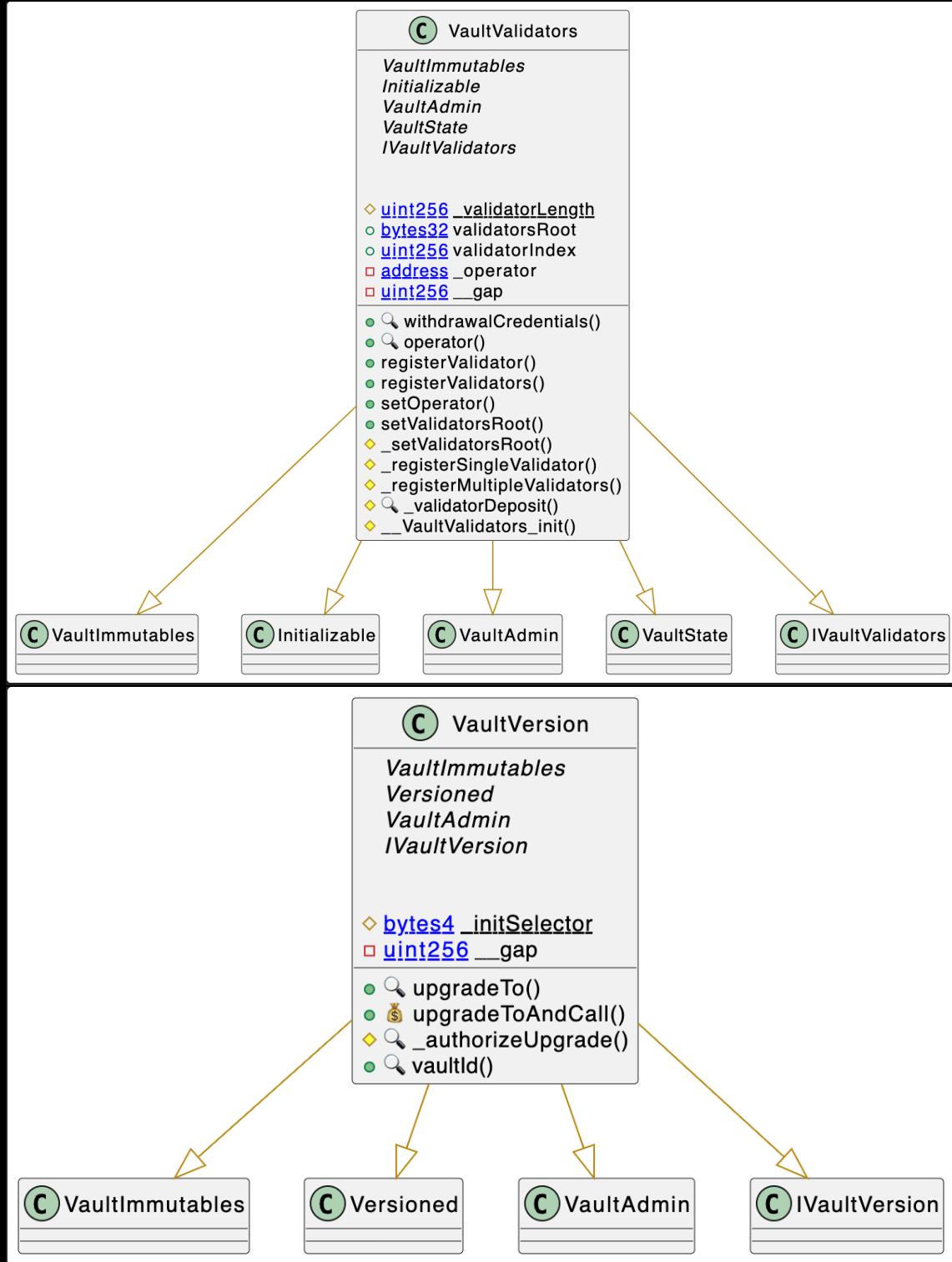
CONTRACT UPGRADABILITY

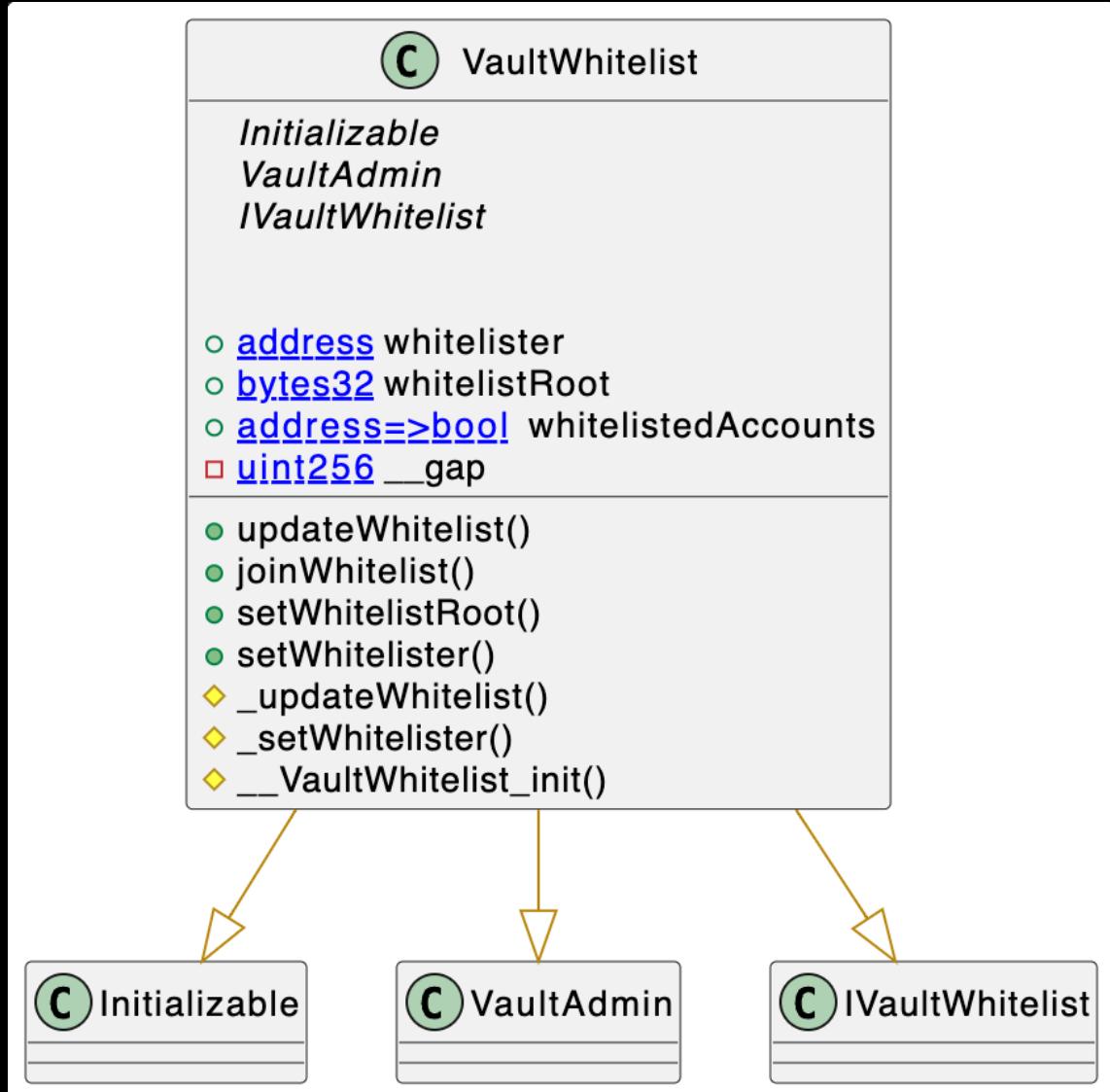


CONTRACT UPGRADABILITY



CONTRACT UPGRADABILITY





6.2 Storage

No possibility of storage collision between the proxy and implementation was identified. The StakeWise team is using the standard `UUPSUpgradeable` along with `ERC1967Upgrade`.

The `UUPSUpgradeable` inherits from the `ERC1967UpgradeUpgradeable` which uses separate implementation, admin, and beacon storage slot.

The `ERC1967Proxy` inherits from the `ERC1967Upgrade` which uses separate

implementation, admin, and beacon storage slot.

6.3 Initialization

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

Listing 6: Keeper.sol (Line 41)

```
41   function initialize(address _owner, uint64 _rewardsDelay)
↳ external override initializer {
42     __KeeperRewards_init(_owner, _rewardsDelay);
43 }
```

Listing 7: EthPrivateVault.sol (Line 40)

```
38  function initialize(
39    bytes calldata params
40  ) external payable override(IEthVault, EthVault) initializer {
41    EthVaultInitParams memory initParams = abi.decode(params, (
↳ EthVaultInitParams));
42    __EthVault_init(initParams);
43    // whitelister is initially set to admin address
44    __VaultWhitelist_init(initParams.admin);
45 }
```

Listing 8: EthVault.sol (Line 61)

```
61   function initialize(bytes calldata params) external payable
↳ virtual override initializer {
62     __EthVault_init(abi.decode(params, (EthVaultInitParams)));
63 }
```

All the parent contracts are correctly initialized:

`Keeper`

- `KeeperRewards` [X]
- `Ownable2StepUpgradeable` [X]

- KeeperValidators [X] (no init function)

EthPrivateVault

- VaultWhitelist [X]
- EthVault [X]
- VaultToken [X]
- VaultAdmin [X]
- VaultFee [X]
- VaultValidators [X]
- VaultEthStaking [X]
- ReentrancyGuardUpgradeable [X]
- VaultMev [X]
- VaultVersion [X] (no init function)
- VaultEnterExit [X] (no init function)
- VaultState [X] (no init function)
- VaultImmutables [X] (no init function)

All relevant child contracts implement constructor with `_disableInitializers()`:

Listing 9: Keeper.sol (Line 38)

```

31   constructor(
32     IOracles _oracles,
33     IVaultsRegistry _vaultsRegistry,
34     IValidatorsRegistry _validatorsRegistry,
35     address sharedMevEscrow
36   ) KeeperValidators(_oracles, _vaultsRegistry,
37     _validatorsRegistry, sharedMevEscrow) {
38     // disable initializers for the implementation contract
39     _disableInitializers();
40 }
```

Listing 10: EthPrivateVault.sol (Line 35)

```

30   constructor(
31     address _keeper,
32     address _vaultsRegistry,
33     address _validatorsRegistry,
34     address sharedMevEscrow
```

```
35     ) EthVault(_keeper, _vaultsRegistry, _validatorsRegistry,
↳ sharedMevEscrow) {}
```

Listing 11: EthVault.sol (Line 57)

```
51  constructor(
52    address _keeper,
53    address _vaultsRegistry,
54    address _validatorsRegistry,
55    address sharedMevEscrow
56  ) VaultImmutables(_keeper, _vaultsRegistry, _validatorsRegistry)
↳ VaultMev(sharedMevEscrow) {
57    _disableInitializers();
58 }
```

Also, the parent contract `ERC20Upgradeable` implements a constructor with `_disableInitializers()`:

Listing 12: EthVault.sol (Line 49)

```
47  constructor() {
48    // disable initializers for the implementation contract
49    _disableInitializers();
50    _initialChainId = block.chainid;
51 }
```

6.4 Deployment

The `eth-full-deploy.ts` file contains an incomplete deployment script. However, `fixtures.ts` contains proof of concept of the deployment used within the integration tests.

The `Keeper` solution is deployed by means of the `deployProxy` functionality. The vaults are deployed via the `EthVaultFactory` contract. No instance of front-run possibility was identified.

CONTRACT UPGRADABILITY

```
export const createKeeper = async function (
  owner: Wallet,
  oracles: Oracles,
  vaultsRegistry: VaultsRegistry,
  validatorsRegistry: Contract,
  sharedMevEscrow: SharedMevEscrow
): Promise<Keeper> {
  const factory = await ethers.getContractFactory('Keeper')
  const instance = await upgrades.deployProxy(factory, [owner.address, REWARDS_DELAY], {
    unsafeAllow: ['delegatecall'],
    constructorArgs: [
      oracles.address,
      vaultsRegistry.address,
      validatorsRegistry.address,
      sharedMevEscrow.address,
    ],
  })
  return (await instance.deployed()) as Keeper
}
```

CONTRACT UPGRADABILITY

```
export const createEthVaultFactory = async function (
  keeper: Keeper,
  vaultsRegistry: VaultsRegistry,
  sharedMevEscrow: SharedMevEscrow,
  validatorsRegistry: Contract
): Promise<EthVaultFactory> {
  const ethVault = await ethers.getContractFactory('EthVault')
  const ethVaultImpl = await upgrades.deployImplementation(ethVault, {
    unsafeAllow: ['delegatecall'],
    constructorArgs: [
      keeper.address,
      vaultsRegistry.address,
      validatorsRegistry.address,
      sharedMevEscrow.address,
    ],
  })

  const ethPrivateVault = await ethers.getContractFactory('EthPrivateVault')
  const ethPrivateVaultImpl = await upgrades.deployImplementation(ethPrivateVault, {
    unsafeAllow: ['delegatecall'],
    constructorArgs: [
      keeper.address,
      vaultsRegistry.address,
      validatorsRegistry.address,
      sharedMevEscrow.address,
    ],
  })

  const factory = await ethers.getContractFactory('EthVaultFactory')
  return (await factory.deploy(
    ethVaultImpl,
    ethPrivateVaultImpl,
    vaultsRegistry.address
  )) as EthVaultFactory
}
```

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:

VaultsRegistry.sol

```
VaultsRegistry.constructor(address)._owner (vaults/VaultsRegistry.sol#28) shadows:
    - Ownable._owner (../node_modules/@openzeppelin/contracts/access/Ownable.sol#21) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

EthVaultFactory.sol

```
OwnMevEscrow.harvest() (vaults/ethereum/mev/OwnMevEscrow.sol#22-31) uses a dangerous strict equality:
    - assets == 0 (vaults/ethereum/mev/OwnMevEscrow.sol#26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

EthVaultFactory.createVault(IEthVaultFactory.VaultParams,bool,bool).mevEscrow (vaults/ethereum/EthVaultFactory.sol#71) is a local variable n
ever initialized
ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool).slot (../node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#92)
is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
EthVaultFactory.constructor(address,address,IVaultsRegistry)._publicVaultImpl (vaults/ethereum/EthVaultFactory.sol#38) lacks a zero-check on
:
    - publicVaultImpl = _publicVaultImpl (vaults/ethereum/EthVaultFactory.sol#39)
EthVaultFactory.constructor(address,address,IVaultsRegistry)._privateVaultImpl (vaults/ethereum/EthVaultFactory.sol#38) lacks a zero-check o
n :
    - privateVaultImpl = _privateVaultImpl (vaults/ethereum/EthVaultFactory.sol#40)
OwnMevEscrow.constructor(address).vault (vaults/ethereum/mev/OwnMevEscrow.sol#17) lacks a zero-check on :
    - vault = address(vault) (vaults/ethereum/mev/OwnMevEscrow.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in EthVaultFactory.createVault(IEthVaultFactory.VaultParams,bool,bool) (vaults/ethereum/EthVaultFactory.sol#52-108):
    External calls:
        - vault = address(new ERC1967Proxy(_privateVaultImpl)) (vaults/ethereum/EthVaultFactory.sol#64)
        - vault = address(new ERC1967Proxy(_publicVaultImpl)) (vaults/ethereum/EthVaultFactory.sol#66)
        - IEthVault(vault).initialize{value: msg.value}{abi.encode(IEthVault.EthVaultInitParams(params.capacity,params.validatorsRoot,msg.se
nder,mevEscrow,params.feePercent,params.name,params.symbol,params.metadataIpfsHash))} (vaults/ethereum/EthVaultFactory.sol#77-90)
        - _vaultsRegistry.addVault(vault) (vaults/ethereum/EthVaultFactory.sol#93)
    External calls sending eth:
        - IEthVault(vault).initialize{value: msg.value}{abi.encode(IEthVault.EthVaultInitParams(params.capacity,params.validatorsRoot,msg.se
nder,mevEscrow,params.feePercent,params.name,params.symbol,params.metadataIpfsHash))} (vaults/ethereum/EthVaultFactory.sol#77-90)
        Event emitted after the call(s):
            - VaultCreated(msg.sender,vault,isPrivate,mevEscrow,params.capacity,params.feePercent,params.name,params.symbol) (vaults/ethereum/Et
hVaultFactory.sol#98-107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

EthVaultFactory.constructor(address,address,IVaultsRegistry) (vaults/ethereum/EthVaultFactory.sol#38-49) uses literals with too many digits:
    - _publicVaultCreateHash = keccak256(bytes)(abi.encodePacked(type()((ERC1967Proxy).creationCode,abi.encode(_publicVaultImpl))) (va
lts/ethereum/EthVaultFactory.sol#43-46)
EthVaultFactory.constructor(address,address,IVaultsRegistry) (vaults/ethereum/EthVaultFactory.sol#38-49) uses literals with too many digits:
    - _privateVaultCreateHash = keccak256(bytes)(abi.encodePacked(type()((ERC1967Proxy).creationCode,abi.encode(_privateVaultImpl))) (va
lts/ethereum/EthVaultFactory.sol#46-48)
EthVaultFactory.computeAddresses(address,bool) (vaults/ethereum/EthVaultFactory.sol#111-125) uses literals with too many digits:
    - ownMevEscrow = Create2.computeAddress(nonce,keccak256(bytes)(abi.encodePacked(type()((OwnMevEscrow).creationCode,abi.encode(vault))
)) (vaults/ethereum/EthVaultFactory.sol#121-124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

EthPrivateVault.sol

```

ERC1967Upgradeable._functionDelegateCall(address,bytes) (./node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967Upgr
radeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function id
  - (success,returnData) = target.delegatecall(data) (./node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967Upgrade
Upgradeable.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
VaultValidators.registerValidators(IKeeperValidators.ApprovalParams,uint256[],bool[],bytes32[]) (vaults/modules/VaultValidators.sol#93-136)
performs a multiplication on the result of a division:
  - validatorsCount = keeperParams.validators.length / _validatorLength (vaults/modules/VaultValidators.sol#103)
  - withdrawableAssets() < _validatorDeposit() * validatorsCount (vaults/modules/VaultValidators.sol#104)
VaultValidators.registerValidators(IKeeperValidators.ApprovalParams,uint256[],bool[],bytes32[]) (vaults/modules/VaultValidators.sol#93-136)
performs a multiplication on the result of a division:
  - validatorsCount = keeperParams.validators.length / _validatorLength (vaults/modules/VaultValidators.sol#103)
  - validatorsCount == 0 || validatorsCount * _validatorLength != keeperParams.validators.length || indexes.length != validatorsCount
(vaults/modules/VaultValidators.sol#111-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
VaultToken._convertToAssets(uint256,uint256,uint256,Math.Rounding) (vaults/modules/VaultToken.sol#90-97) uses a dangerous strict equality:
  - (totalShares == 0) (vaults/modules/VaultToken.sol#96)
VaultToken._convertToShares(uint256,uint256,uint256,Math.Rounding) (vaults/modules/VaultToken.sol#73-85) uses a dangerous strict equality:
  - (assets == 0 || totalShares == 0) (vaults/modules/VaultToken.sol#81-84)
VaultState._processTotalAssetsDelta(int256) (vaults/modules/VaultState.sol#74-125) uses a dangerous strict equality:
  - feeRecipientShares == 0 (vaults/modules/VaultState.sol#111)
VaultState._updateExitQueue() (vaults/modules/VaultState.sol#130-165) uses a dangerous strict equality:
  - _queuedShares == 0 (vaults/modules/VaultState.sol#133)
VaultState._updateExitQueue() (vaults/modules/VaultState.sol#130-165) uses a dangerous strict equality:
  - exitedAssets == 0 (vaults/modules/VaultState.sol#141)
VaultState._updateExitQueue() (vaults/modules/VaultState.sol#130-165) uses a dangerous strict equality:
  - burnedShares == 0 (vaults/modules/VaultState.sol#145)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
Reentrancy in VaultEthStaking.updateStateAndDeposit(address,address,IKeeperRewards.HarvestParams) (vaults/modules/VaultEthStaking.sol#47-54)
:
  External calls:
  - updateState(harvestParams) (vaults/modules/VaultEthStaking.sol#52)
    - (totalAssetsDelta,unlockedMevDelta) = IKeeperRewards(keeper).harvest(harvestParams) (vaults/modules/VaultMev.sol#44-46)
    - ISharedMevEscrow(_mevEscrow).harvest(unlockedMevDelta) (vaults/modules/VaultMev.sol#53)
    - totalAssetsDelta + int256(IOwnMevEscrow(_mevEscrow).harvest()) (vaults/modules/VaultMev.sol#59)
State variables written after the call(s):
  - deposit(receiver,refferrer) (vaults/modules/VaultEthStaking.sol#53)
    - _totalAssets = SafeCast.toInt128(totalAssetsAfter) (vaults/modules/VaultEnterExit.sol#147)
VaultToken._totalAssets (vaults/modules/VaultToken.sol#21) can be used in cross function reentrances:
  - VaultEnterExit._deposit(address,uint256,address) (vaults/modules/VaultEnterExit.sol#127-157)
  - VaultState._processTotalAssetsDelta(int256) (vaults/modules/VaultState.sol#74-125)
  - VaultState._updateExitQueue() (vaults/modules/VaultState.sol#130-165)
  - VaultEnterExit._withdraw(address,address,uint256,uint256) (vaults/modules/VaultEnterExit.sol#166-191)
  - VaultToken.convertToAssets(uint256) (vaults/modules/VaultToken.sol#50-52)
  - VaultToken.convertToShares(uint256) (vaults/modules/VaultToken.sol#45-47)
  - VaultToken.totalAssets() (vaults/modules/VaultToken.sol#40-42)
  - VaultEnterExit.withdraw(uint256,address,address) (vaults/modules/VaultEnterExit.sol#23-31)
  - deposit(receiver,refferrer) (vaults/modules/VaultEthStaking.sol#53)
    - _totalShares += SafeCast.toInt128(shares) (vaults/modules/VaultEnterExit.sol#146)
VaultToken._totalShares (vaults/modules/VaultToken.sol#20) can be used in cross function reentrances:
  - VaultEnterExit._deposit(address,uint256,address) (vaults/modules/VaultEnterExit.sol#127-157)
  - VaultState._processTotalAssetsDelta(int256) (vaults/modules/VaultState.sol#74-125)
  - VaultState._updateExitQueue() (vaults/modules/VaultState.sol#130-165)
  - VaultEnterExit._withdraw(address,address,uint256,uint256) (vaults/modules/VaultEnterExit.sol#166-191)
  - VaultToken.convertToAssets(uint256) (vaults/modules/VaultToken.sol#50-52)
  - VaultToken.convertToShares(uint256) (vaults/modules/VaultToken.sol#45-47)
  - VaultToken.totalSupply() (vaults/modules/VaultToken.sol#35-37)
  - VaultEnterExit.withdraw(uint256,address,address) (vaults/modules/VaultEnterExit.sol#23-31)
  - deposit(receiver,refferrer) (vaults/modules/VaultEthStaking.sol#53)
    - balanceOf[0] += shares (vaults/modules/VaultEnterExit.sol#152)
ERC20Upgradeable.balanceOf (base/ERC20upgradeable.sol#28) can be used in cross function reentrances:
  - VaultEnterExit._deposit(address,uint256,address) (vaults/modules/VaultEnterExit.sol#127-157)
  - VaultState._processTotalAssetsDelta(int256) (vaults/modules/VaultState.sol#74-125)
  - ERC20Upgradeable._transfer(address,address,uint256) (base/ERC20Upgradeable.sol#142-153)
  - VaultEnterExit._withdraw(address,address,uint256,uint256) (vaults/modules/VaultEnterExit.sol#166-191)
  - ERC20Upgradeable.balanceOf (base/ERC20Upgradeable.sol#28)
  - VaultEnterExit.enterExitQueue(uint256,address,address) (vaults/modules/VaultEnterExit.sol#45-73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
Multicall.multicall(bytes[]) (base/Multicall.sol#15-31) has external calls inside a loop: (success,result) = address(this).delegatecall(data
[i]) (base/Multicall.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-inside-a-loop
Reentrancy in VaultValidators.registerValidator(IKeeperValidators.ApprovalParams,bytes32[]) (vaults/modules/VaultValidators.sol#55-90):
  External calls:
  - IKeeperValidators(keeper).approveValidators(keeperParams) (vaults/modules/VaultValidators.sol#60)
State variables written after the call(s):
  - validatorIndex = currentIndex + 1 (vaults/modules/VaultValidators.sol#88)
Reentrancy in VaultValidators.registerValidators(IKeeperValidators.ApprovalParams,uint256[],bool[],bytes32[]) (vaults/modules/VaultValidator
s.sol#93-136):
  External calls:
  - IKeeperValidators(keeper).approveValidators(keeperParams) (vaults/modules/VaultValidators.sol#100)
State variables written after the call(s):
  - validatorIndex += validatorsCount (vaults/modules/VaultValidators.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

```

```

Reentrancy in VaultEthStaking._registerMultipleValidators(bytes,uint256[]) (vaults/modules/VaultEthStaking.sol#82-121):
    External calls:
        - IEthValidatorsRegistry(validatorsRegistry).deposit{value: validatorDeposit}({publicKey,withdrawalCreds,validator,bytes32(serializer)})
    ) (vaults/modules/VaultEthStaking.sol#107-112)
        Event emitted after the call(s):
            - ValidatorRegistered(publicKey) (vaults/modules/VaultEthStaking.sol#119)
Reentrancy in VaultEthStaking._registerSingleValidator(bytes) (vaults/modules/VaultEthStaking.sol#69-79):
    External calls:
        - IEthValidatorsRegistry(validatorsRegistry).deposit{value: _validatorDeposit()}({publicKey,withdrawalCredentials(),validator,bytes32(serializer)})
    ) (vaults/modules/VaultEthStaking.sol#71-76)
        Event emitted after the call(s):
            - ValidatorRegistered(publicKey) (vaults/modules/VaultEthStaking.sol#78)
Reentrancy in VaultEthStaking.updateStateAndDeposit(address,address,IKeeperRewards.HarvestParams) (vaults/modules/VaultEthStaking.sol#47-54):
:
    External calls:
        - updateState(harvestParams) (vaults/modules/VaultEthStaking.sol#52)
            - (totalAssetsDelta,unlockedMevDelta) = IKeeperRewards(keeper).harvest(harvestParams) (vaults/modules/VaultMev.sol#44-46)
            - ISharedMevEscrow(_mevEscrow).harvest(unlockedMevDelta) (vaults/modules/VaultMev.sol#53)
            - totalAssetsDelta + int256(IOwnMevEscrow(_mevEscrow).harvest()) (vaults/modules/VaultMev.sol#59)
    Event emitted after the call(s):
        - Deposit(msg.sender,to,assets,shares,referrer) (vaults/modules/VaultEnterExit.sol#156)
            - deposit(receiver,referrer) (vaults/modules/VaultEnterExit.sol#53)
        - Transfer(address(0),to,shares) (vaults/modules/VaultEnterExit.sol#155)
            - deposit(receiver,referrer) (vaults/modules/VaultEnterExit.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Variable VaultToken._totalAssets (vaults/modules/VaultToken.sol#21) is too similar to VaultToken._convertToShares(uint256,uint256,uint256,Math.Rounding).totalAssets_ (vaults/modules/VaultToken.sol#76)
Variable VaultToken._totalAssets (vaults/modules/VaultToken.sol#21) is too similar to VaultToken._convertToAssets(uint256,uint256,uint256,Math.Rounding).totalAssets_ (vaults/modules/VaultToken.sol#93)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

```

Keeper.sol

```

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

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

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (../node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID() (../node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Keeper.initialize(address,uint64) .._owner (keeper/Keeper.sol#42) shadows:
    - OwnableUpgradeable.._owner (../node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#22) (state variable)
KeeperRewards.__KeeperRewards_init(address,uint64) .._owner (keeper/KeeperRewards.sol#238) shadows:
    - OwnableUpgradeable.._owner (../node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#22) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

KeeperRewards.canUpdateRewards() (keeper/KeeperRewards.sol#117-124) uses timestamp for comparisons
    Dangerous comparisons:
        - _lastRewardsTimestamp + rewardsDelay < block.timestamp (keeper/KeeperRewards.sol#122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Function IValidatorsRegistry.get_deposit_root() (interfaces/IValidatorsRegistry.sol#22) is not in mixedCase
Parameter Keeper.initialize(address,uint64).._owner (keeper/Keeper.sol#42) is not in mixedCase
Parameter Keeper.initialize(address,uint64).._rewardsDelay (keeper/Keeper.sol#42) is not in mixedCase
Parameter KeeperRewards.setRewardsDelay(uint64).._rewardsDelay (keeper/KeeperRewards.sol#207) is not in mixedCase
Function KeeperRewards.__KeeperRewards_init(address,uint64) (keeper/KeeperRewards.sol#238-245) is not in mixedCase
Parameter KeeperRewards.__KeeperRewards_init(address,uint64).._owner (keeper/KeeperRewards.sol#238) is not in mixedCase
Parameter KeeperRewards.__KeeperRewards_init(address,uint64).._rewardsDelay (keeper/KeeperRewards.sol#238) is not in mixedCase
Constant KeeperRewards.._rewardsRootTypeHash (keeper/KeeperRewards.sol#20-23) is not in UPPER_CASE_WITH_UNDERSCORES
Variable KeeperRewards..__gap (keeper/KeeperRewards.sol#252) is not in mixedCase
Constant KeeperValidators..registerValidatorsTypeHash (keeper/KeeperValidators.sol#18-21) is not in UPPER_CASE_WITH_UNDERSCORES
Constant KeeperValidators..updateExitSigTypeHash (keeper/KeeperValidators.sol#23-24) is not in UPPER_CASE_WITH_UNDERSCORES
Variable KeeperValidators..__gap (keeper/KeeperValidators.sol#115) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

```

OwnMevEscrow.sol

```

OwnMevEscrow.harvest() (vaults/ethereum/mev/OwnMevEscrow.sol#22-31) uses a dangerous strict equality:
    - assets == 0 (vaults/ethereum/mev/OwnMevEscrow.sol#26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

OwnMevEscrow.constructor(address).._vault (vaults/ethereum/mev/OwnMevEscrow.sol#17) lacks a zero-check on :
    - vault = address(_vault) (vaults/ethereum/mev/OwnMevEscrow.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

```

- Re-entrancy issues are false positives.
- Usage of timestamp for comparisons is false positive.

- Vast number of findings are false positives.
- Findings related to third-party libraries (OpenZeppelin) were omitted.
- Findings related to the different versions of Solidity usage and Pragma versions were omitted.
- The scan of `SharedMevEscrow.sol` and `Oracles.sol` yielded no result.
- The scan of `EthVault.sol` is included in the scan of `EthPrivateVault.sol`
- No major issues were found by Slither.

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:

EthPrivateVault.sol

Report for contracts/base/Multicall.sol
<https://dashboard.mythx.io/#/console/analyses/2f176cda-7efd-4b47-bff4-d9e815edce46>

Line	SWC Title	Severity	Short Description
15	(SWC-118) Incorrect Constructor Name	Medium	Potential incorrect constructor name "multicall".

Oracles.sol

Report for contracts/keeper/Oracles.sol
<https://dashboard.mythx.io/#/console/analyses/1d10abee-4a80-4acd-9f9b-26392c0753ce>

Line	SWC Title	Severity	Short Description
41	(SWC-110) Assert Violation	Unknown	Out of bounds array access
43	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
58	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
71	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
71	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
117	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
127	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
137	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
138	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered

- The scan of EthVault.sol is included in the scan of EthPrivateVault.sol
- The scan of VaultsRegistry.sol, EthVaultFactory.sol.sol, OwnMevEscrow.sol, SharedMevEscrow.sol, and Keeper.sol yielded no result.

AUTOMATED TESTING

- No major issues were found by Mythx.

THANK YOU FOR CHOOSING
 HALBORN