Alluvial - Liquid Collective

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 1st, 2022 - August 2nd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/02/2022	Ataberk Yavuzer
0.2	Document Edits	08/02/2022	Ataberk Yavuzer
0.3	Draft Review	08/03/2022	Gabi Urrutia
1.0	Remediation Plan	08/11/2022	Ataberk Yavuzer
1.1	Remediation Plan Review	08/11/2022	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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Alluvial engaged Halborn to conduct a security audit on their smart contracts beginning on July 1st, 2022 and ending on August 2nd, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one month for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Alluvial team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of 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.
- Dynamic Analysis (foundry)
- Static Analysis(slither)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Liquid Collective Contracts Security Audit Test Scope
   (a) Repository: River Contracts
   (b) Commit ID: 6c2bef46955b2e38dfebc7e135ee86b616fcbcb9
2. Out-of-Scope
   (a) contracts/src/mock/*.sol
   (b) contracts/test/*.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	4	5	5

LIKELIHOOD

IMPACT

		(HAL-02)	(HAL-01)	
		(HAL-03) (HAL-04) (HAL-05)		
	(HAL-08) (HAL-09) (HAL-10)	(HAL-06) (HAL-07)		
(HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) DONATE CALL BEFORE DEPOSIT LEADS LOSS OF POSSIBLE REWARDS	High	SOLVED - 08/10/2022
(HAL-02) ORACLE SHOULD CHECK UNDERLYING BALANCE INSTEAD OF TOTAL SUPPLY	Medium	SOLVED - 08/10/2022
(HAL-03) DIVISION BY ZERO	Medium	SOLVED - 08/10/2022
(HAL-04) MALICIOUS OWNER CAN ADD AN OPERATOR WITH EXISTING NAME	Medium	SOLVED - 08/10/2022
(HAL-05) SINGLE-STEP OWNERSHIP CHANGE	Medium	SOLVED - 08/10/2022
(HAL-06) ACCIDENTALLY SENT ETHERS WILL GET STUCK IN PROTOCOL	Low	RISK ACCEPTED
(HAL-07) MISSING REENTRANCY GUARD	Low	SOLVED - 08/10/2022
(HAL-08) IGNORED RETURN VALUES	Low	SOLVED - 08/10/2022
(HAL-09) LACK OF ZERO ADDRESS CHECKS	Low	SOLVED - 08/10/2022
(HAL-10) USE OF UNNECESSARY IFADMIN MODIFIER	Low	NOT APPLICABLE
(HAL-11) USE UNCHECKED KEYWORD FOR GAS OPTIMISATION	Informational	SOLVED - 08/10/2022
(HAL-12) USE OF POST-FIX INCREMENT ON FOR LOOPS	Informational	SOLVED - 08/10/2022
(HAL-13) UNNECESSARY ASSERT USAGE	Informational	SOLVED - 08/10/2022
(HAL-14) IF CONDITIONS CAN BE OPTIMISED	Informational	SOLVED - 08/10/2022
(HAL-15) UNNECESSARY ADDITION OF TRANSFER AND DEPOSIT MASKS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) DONATE CALL BEFORE DEPOSIT LEADS LOSS OF POSSIBLE REWARDS - HIGH

Description:

If the totalSupply() is zero for the Alluvial, the donate() function will not mint any rewards.

The documentation explains the use of donate function as below:

> Allows anyone to add ethers to river without minting new shares

However, if the totalSupply() exceeds zero, the contract tries to mint rewards for operators according to the _onEarnings function. Therefore, the proposition above can be ignored for this case, since the donate function calls _onDonation and _onEarnings functions in order.

The sharesToMint variable will always return zero if the contract does not have any LsETH. There should be a sanity check on the contract to prevent the donating operation while totalSupply() is zero.

For example:
Scenario-1: Bob donates **10 ETH** while totalSupply is zero. The contract mints 0 River as reward.
Scenario-2: Bob donates **10 ETH** while totalSupply is 1e18. The contract transfers **47619047619047619** River to operators as reward.

PoC - Foundry Test Case:

Listing 1: testDonateBeforeDeposit

```
function testDonateBeforeDeposit() public {
        vm.prank(admin);
        river.setGlobalFee(5000);
        vm.startPrank(bob);
        river.donate{value: 10 ether}(); // returns 0
```

Listing 2: testDonateAfterDeposit

```
function testDonateAfterDeposit() public {
        vm.prank(admin);
        river.setGlobalFee(5000);
        vm.startPrank(bob);
        river.deposit{value: 1 ether}();
        river.donate{value: 10 ether}(); // returns
↳ 47619047619047619
```

Code Location:

Listing 3: River.1.sol (Lines 194,201)

```
192 function _onEarnings(uint256 _amount) internal override {
193     uint256 globalFee = GlobalFee.get();
194     uint256 sharesToMint = (_amount * _totalSupply() * globalFee)
↳ /
195         ((_assetBalance() * BASE) - (_amount * globalFee));
196
197     uint256 operatorRewards = (sharesToMint * OperatorRewardsShare
↳ .get()) / BASE;
198
199     uint256 mintedRewards = _rewardOperators(operatorRewards);
200
201     _mintRawShares(TreasuryAddress.get(), sharesToMint -
↳ mintedRewards);
202 }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to add a sanity check for `_onEarnings` function to prevent donating if the `totalSupply` is zero.

Remediation Plan:

SOLVED: This issue was solved by adding another control to the contract.

Commit ID: 6a46a1b47edaa6bc90f6c269011be835dec5c341

3.2 (HAL-02) ORACLE SHOULD CHECK UNDERLYING BALANCE INSTEAD OF TOTAL SUPPLY - MEDIUM

Description:

The reportBeacon function reports the beacon data to the protocol to correctly validate price of underlying asset. This function also makes an internal call to the _pushToRiver function. The contract tries to calculate the ETH balance after and before during the _pushToRiver call. However, it uses totalSupply instead of the current balance, which is the underlying balance.

Using total supply instead of underlying balance will lead to confusion about asset prices. The price of the asset may differ negatively, as the oracle will send less or more than the expected price.

Code Location:

Listing 4: Oracle.1.sol (Lines 492,494)

```
483 function _pushToRiver(
484     uint256 _epochId,
485     uint128 _balanceSum,
486     uint32 _validatorCount,
487     BeaconSpec.BeaconSpecStruct memory _beaconSpec
488 ) internal {
489     _clearReporting(_epochId + _beaconSpec.epochsPerFrame);
490
491     IRiverOracleInput riverAddress = IRiverOracleInput(
 ↳ RiverAddress.get());
492     uint256 prevTotalEth = riverAddress.totalSupply();
493     riverAddress.setBeaconData(_validatorCount, _balanceSum,
 ↳ bytes32(_epochId));
494     uint256 postTotalEth = riverAddress.totalSupply();
495
496     uint256 timeElapsed = (_epochId - LastEpochId.get()) *
 ↳ _beaconSpec.slotsPerEpoch * _beaconSpec.secondsPerSlot;
```

```
497
498     _sanityChecks(postTotalEth, prevTotalEth, timeElapsed);
499     LastEpochId.set(_epochId);
500
501     emit PostTotalShares(postTotalEth, prevTotalEth, timeElapsed,
 ↳ riverAddress.totalShares());
502 }
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Use the totalUnderlyingSupply() function for the total ETH calculation in the _pushToRiver function instead of the totalSupply() function.

Remediation Plan:

SOLVED: This issue was solved in the following commit:

Commit ID: 4b4ef76c93e215ceb1218d68f73a833c766fa134

3.3 (HAL-03) DIVISION BY ZERO - MEDIUM

Description:

During the audit, it was determined that there is a “Division by Zero” result in the _onEarnings function. In Solidity, when you try to set any value to zero, the execution will be reverted with an error message. Therefore, the transaction will also be reverted.

The sharesToMint variable tries to divide (_amount * _totalSupply()* globalFee) to ((_assetBalance()* BASE)- (_amount * globalFee)).

If (_assetBalance()* BASE) equals _amount * globalFee, the denominator will be zero. As a result, division by zero will occur.

Code Location:

Listing 5: River.1.sol (Lines 194,195)

```
192 function _onEarnings(uint256 _amount) internal override {
193     uint256 globalFee = GlobalFee.get();
194     uint256 sharesToMint = (_amount * _totalSupply() * globalFee)
 ↳ /
195         ((_assetBalance() * BASE) - (_amount * globalFee));
196
197     uint256 operatorRewards = (sharesToMint * OperatorRewardsShare
 ↳ .get()) / BASE;
198
199     uint256 mintedRewards = _rewardOperators(operatorRewards);
200
201     _mintRawShares(TreasuryAddress.get(), sharesToMint -
 ↳ mintedRewards);
202 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is suggested to add a check to verify if `(_assetBalance()* BASE)` has the same value as `(_amount * globalFee)`. In this case, the result should return zero.

Remediation Plan:

SOLVED: This issue was solved by adding another denominator check to the contract.

Commit ID: 799c72d45441d6a3a0828a381149a96611ea656e

3.4 (HAL-04) MALICIOUS OWNER CAN ADD AN OPERATOR WITH SAME ADDRESS - MEDIUM

Description:

It is possible to add the same operator to the contract using a different name. In this case, the contract will have two identical operators. During the donate event, the contract sends rewards to these operators. If the number of identical operators increases, other operators will get fewer rewards.

For example,

- The contract has 2 Operators: Operator-1 and Operator-2.
- _rewardOperators sends 100 RIVER in total to both operators.
- Operator-1 gets 50 RIVER and Operator-2 gets 50 RIVER.
- The malicious owner adds another operator to the contract that has the same address as Operator-2.
- _rewardOperators sends 100 RIVER again.
- rewardsPerActiveValidator returns 33 since there are three operators in the contract.
- Operator-1 gets 33 RIVER and Operator-2 gets 66 RIVER since the new operator has the same address as Operator-2.

Code Location:

Listing 6: OperatorsManager.1.sol (Lines 103-106)

```
 99 function addOperator(
100     string calldata _name,
101     address _operator,
102     address _feeRecipient
103 ) external onlyAdmin {
104     if (Operators.exists(_name) == true) {
105         revert OperatorAlreadyExists(_name);
106     }
107
108     Operators.Operator memory newOperator = Operators.Operator({
109         active: true,
110         operator: _operator,
111         feeRecipient: _feeRecipient,
112         name: _name,
113         limit: 0,
114         funded: 0,
115         keys: 0,
116         stopped: 0
117     });
118
119     uint256 operatorIndex = Operators.set(_name, newOperator);
120
121     emit AddedOperator(operatorIndex, newOperator.name,
 ↳ newOperator.operator, newOperator.feeRecipient);
122 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

The sanity check in the addOperator function only checks that if the operator name already exists in the contract. This function should also check if the operator address already exists.

Remediation Plan:

SOLVED: It is now not possible to add an existing operator to the contract. The issue was solved in the following commit:

Commit ID: 398ddd6363f93563fa076d52026bb45d06d1a485

3.5 (HAL-05) SINGLE-STEP OWNERSHIP CHANGE - MEDIUM

Description:

Single-step ownership change for contracts is risky. Owner addresses in River contracts can be changed in one step due to pattern in LibOwnable. If the owner's address is set to the wrong address, this could lead to funds being lost or locked.

When changing privileged roles, a two-step approach is recommended:

1. The current privileged role proposes a new address for change
2. The proposed new address then claims the privileged role in a separate transaction.

Code Location:

Listing 7: River.1.sol

```
113 function setAdministrator(address _newAdmin) external onlyAdmin {
114     LibOwnable._setAdmin(_newAdmin);
115 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended that you implement a two-step ownership change instead of a one-step ownership change.

Remediation Plan:

SOLVED: This issue was solved by implementing the two-step ownership change in the contract.

Commit ID: 2c3fdc8d2fa91c045d4d7332d81ee044b7e8f3da

3.6 (HAL-06) ACCIDENTALLY SENT ETHERS WILL GET STUCK IN PROTOCOL - LOW

Description:

Allowed users can deposit or donate ETH to the River contract. If these users accidentally send ETH to this contract, there is no way to revert this error. River's contract has no withdraw function. Therefore, users will not be able to retrieve their accidentally sent ETH.

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to have a withdraw feature in the contract for these types of cases.

Remediation Plan:

RISK ACCEPTED: The risk of this finding was accepted.

> Currently, this is intended behavior. The withdrawal process will be defined and implemented in details once the spec is written in stone. As we advance with unknowns, we currently have a stub withdrawal contract ready to accept all the exited funds, where the implementation will be changed to manage all this process.

3.7 (HAL-07) MISSING REENTRANCY GUARD - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Code Location:

Listing 8: WLSETH.1.sol (Line 124)

```
124 function mint(address _recipient, uint256 _value) external {
125     BalanceOf.set(_recipient, BalanceOf.get(_recipient) + _value);
126     IRiverToken(RiverAddress.get()).transferFrom(msg.sender,
 ↳ address(this), _value);
127 }
```

Listing 9: WLSETH.1.sol (Line 134)

```
134 function burn(address _recipient, uint256 _value) external {
135     uint256 callerUnderlyingBalance = IRiverToken(RiverAddress.get
 ↳ ()).underlyingBalanceFromShares(
136         BalanceOf.get(msg.sender)
137     );
138     if (_value > callerUnderlyingBalance) {
139         revert BalanceTooLow();
140     }
141     uint256 sharesAmount = IRiverToken(RiverAddress.get()).
 ↳ sharesFromUnderlyingBalance(_value);
142     BalanceOf.set(msg.sender, BalanceOf.get(msg.sender) -
 ↳ sharesAmount);
143     IRiverToken(RiverAddress.get()).transfer(_recipient,
 ↳ sharesAmount);
144 }
```

Listing 10: TransferManager.1.sol (Line 49)

```
49 function deposit() external payable {
50     _deposit(msg.sender);
51 }
```

Listing 11: TransferManager.1.sol (Line 61)

```
61 function donate() external payable {
62     if (msg.value == 0) {
63         revert EmptyDonation();
64     }
65
66     _onDonation(msg.value);
67
68     emit Donation(msg.sender, msg.value);
69 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

The functions in the code location section are missing nonReentrant modifiers. It is recommended to add the OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation Plan:

SOLVED: This finding was solved for the `burn()` and `mint()` methods after implementing `nonReentrant` modifier for these methods.

Commit ID: e9be5d57d67568b6e47ee0748945a4884c939a4d

3.8 (HAL-08) IGNORED RETURN VALUES - LOW

Description:

The transferFrom and transfer functions are declared to return a boolean return variable after successful transfers. However, it does not return any variables during calls to the WLSETH.mint() and WLSETH.burn() functions. It is important to validate these return variables. In this case, calling these functions can break any integrations or composability.

Code Location:

Listing 12: WLSETH.1.sol (Line 126)

```
124 function mint(address _recipient, uint256 _value) external {
125     BalanceOf.set(_recipient, BalanceOf.get(_recipient) + _value);
126     IRiverToken(RiverAddress.get()).transferFrom(msg.sender,
 ↳ address(this), _value);
127 }
```

Listing 13: WLSETH.1.sol (Line 143)

```
134 function burn(address _recipient, uint256 _value) external {
135     uint256 callerUnderlyingBalance = IRiverToken(RiverAddress.get
 ↳ ()).underlyingBalanceFromShares(
136         BalanceOf.get(msg.sender)
137     );
138     if (_value > callerUnderlyingBalance) {
139         revert BalanceTooLow();
140     }
141     uint256 sharesAmount = IRiverToken(RiverAddress.get()).
 ↳ sharesFromUnderlyingBalance(_value);
142     BalanceOf.set(msg.sender, BalanceOf.get(msg.sender) -
 ↳ sharesAmount);
143     IRiverToken(RiverAddress.get()).transfer(_recipient,
 ↳ sharesAmount);
144 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is not recommended to ignore these return variables. These boolean values should be returned during the function calls.

Remediation Plan:

SOLVED: This vulnerability was resolved by Alluvial team after adding additional code that checks the return value from transferFrom() method.

Commit ID: cab51608d19a44e0c165715bc6e7a970d657b19a

3.9 (HAL-09) LACK OF ZERO ADDRESS CHECKS - LOW

Description:

River Contracts have address fields in multiple functions. These functions are missing address validations. Each address should be validated and checked to be non-zero. This is also considered a best practice.

During testing, it has been found that some of these inputs are not protected against using address(0) as the destination address.

Code Location:

Listing 14: Functions with missing zero address checks

```
OracleManagerV1.initOracleManagerV1::_oracle
OperatorsManagerV1.addOperator::_operator,_feeRecipient
DepositManagerV1.initDepositManagerV1::_depositContractAddress
WLSETHV1.mint::_recipient
WLSETHV1.burn::_recipient
RiverV1.setAdministrator::_newAdmin
RiverV1.setTreasury::_newTreasury
RiverV1.setAllowlist::_newAllowlist
Firewall.constructor::governor_,executor_
Firewall.changeGovernor::newGovernor
Firewall.changeExecutor::newExecutor
ELFeeRecipientV1.initELFeeRecipientV1::_riverAddress
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to validate that each address input is non-zero.

Remediation Plan:

SOLVED: The issue was solved by adding zero address checks.

Commit ID: 4a526b0a9f82537bdf582fee1475171433149216

3.10 (HAL-10) USE OF UNNECESSARY IFADMIN MODIFIER - LOW

Description:

The ifAdmin modifier is designed for functions that can be executable by only the admin role. The isPaused function on the TUPProxy contract checks that if the current contract is paused or not. It returns true or false accordingly. However, this function has unnecessary ifAdmin modifier. Therefore, only the contract admin can view the paused state of contract.

Code Location:

Listing 15: TUPProxy.sol (Line 23)

```
23 function isPaused() external ifAdmin returns (bool) {
24     return StorageSlot.getBooleanSlot(_PAUSE_SLOT).value;
25 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Remove the unnecessary ifAdmin modifier from the isPaused function.

Remediation Plan:

NOT APPLICABLE: This finding is not applicable because it was designed as an intended behavior.

> This was intended as the ifAdmin has two roles:
> - Protects sensitive calls from being performed by random actors
> - Prevent method collision between proxy and implementation

> The isPaused case falls in the second case as it prevents any collision with the implementation (we can have an unrelated isPaused method in the implementation and users won't call the one in the Proxy, only the admin will)

3.11 (HAL-11) USE UNCHECKED KEYWORD FOR GAS OPTIMISATION - INFORMATIONAL

Description:

In Solidity (pragma 0.8.0 and later), adding unchecked keyword for arithmetical operations can decrease gas usage on contracts where underflow/underflow is unrealistic. It is possible to save gas by using this keyword on multiple code locations.

Code Location:

Listing 16: Oracle.1.sol (Line 209)

```
207 function isMember(address _memberAddress) external view returns (
 ↳ bool) {
208     address[] memory members = OracleMembers.get();
209     for (uint256 idx = 0; idx < members.length; ++idx) {
210         if (members[idx] == _memberAddress) {
211             return true;
212         }
213     }
214     return false;
215 }
```

Listing 17: River.1.sol (Line 165)

```
165 for (uint256 idx = 0; idx < operators.length; ++idx) {
166     uint256 operatorActiveValidatorCount = operators[idx].funded -
 ↳  operators[idx].stopped;
167     totalActiveValidators += operatorActiveValidatorCount;
168     validatorCounts[idx] = operatorActiveValidatorCount;
169 }
```

Listing 18: River.1.sol (Line 174)

```
174 for (uint256 idx = 0; idx < validatorCounts.length; ++idx) {
175     _mintRawShares(operators[idx].feeRecipient, validatorCounts[
 ↳ idx] * rewardsPerActiveValidator);
176 }
```

Listing 19: DepositManager.1.sol (Line 87)

```
87 for (uint256 idx = 0; idx < receivedPublicKeyCount; idx += 1) {
88     _depositValidator(publicKeys[idx], signatures[idx],
 ↳ withdrawalCredentials);
89 }
```

Listing 20: OperatorsManager.1.sol (Line 235)

```
235 for (uint256 idx = 0; idx < _keyCount; ++idx) {
236     bytes memory publicKey = BytesLib.slice(
237         _publicKeys,
238         idx * ValidatorKeys.PUBLIC_KEY_LENGTH,
239         ValidatorKeys.PUBLIC_KEY_LENGTH
240     );
```

Listing 21: OperatorsManager.1.sol (Line 267)

```
267 for (uint256 idx = 0; idx < _indexes.length; ++idx) {
268     uint256 keyIndex = _indexes[idx];
269     ...
```

Listing 22: OperatorsManager.1.sol (Lines 336,339)

```
336 for (uint256 idx = 0; idx < arr1.length; ++idx) {
337     res[idx] = arr1[idx];
338 }
339 for (uint256 idx = 0; idx < arr2.length; ++idx) {
340     res[idx + arr1.length] = arr2[idx];
341 }
```

Listing 23: OperatorsManager.1.sol (Line 357)

```
357 for (uint256 idx = 1; idx < operators.length; ++idx) {
358     if (
359         operators[idx].funded - operators[idx].stopped <
360         operators[selectedOperatorIndex].funded - operators[
 ↳ selectedOperatorIndex].stopped
361     ) {
362         selectedOperatorIndex = idx;
363     }
364 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to follow the suggestion below to optimize gas usage on for loops.

Listing 24: Optimised for loops

```
1 for (uint256 idx; idx < array.length; ) {
2     // function logic
3     unchecked {
4         ++idx;
5     }
6 }
```

Remediation Plan:

SOLVED: This finding was solved by applying the suggestion above for all `for` loops in the protocol.

Commit ID: 092c468a112955aebd8a5ed2d80507393f9836c8

3.12 (HAL-12) USE OF POST-FIX INCREMENT ON FOR LOOPS - INFORMATIONAL

Description:

In all for loops, the index variable is incremented using +=. It is known that, in loops, using ++i costs less gas per iteration than +=. This also affects incremented variables within the loop code block.

Code Location:

Listing 25: DepositManager.1.sol (Line 87)

```
87 for (uint256 idx = 0; idx < receivedPublicKeyCount; idx += 1) {
88     _depositValidator(publicKeys[idx], signatures[idx],
↳ withdrawalCredentials);
89 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to apply the following pattern for Solidity pragma version 0.8.0 and later.

Listing 26: Possible Fix

```
for (uint256 i = startIdx; i < arrayLength; ) {
    . . .
    unchecked {
     ++i
    }
```

Remediation Plan:

SOLVED: This finding was solved by applying the suggestion above for all `for` loops in the protocol.

Commit ID: 092c468a112955aebd8a5ed2d80507393f9836c8

3.13 (HAL-13) UNNECESSARY ASSERT USAGE - INFORMATIONAL

Description:

The _depositValidator function on the DepositManagerV1 contract tries to calculate depositAmount variable correctly by using assert. It tries to divide the DEPOSIT_SIZE variable into 1000000000 wei. Subsequently, it tries to multiply the depositAmount variable with 1000000000 wei to check this calculation if equals to value variable, which is declared as DEPOSIT_SIZE already.

The assert usage here does not benefit anything to the contract, since value is hardcoded as DEPOSIT_SIZE. The result will always be the same.

Code Location:

Listing 27: DepositManager.1.sol (Lines 112,114,115)

```
112 uint256 value = DEPOSIT_SIZE;
113
114 uint256 depositAmount = value / 1000000000 wei;
115 assert(depositAmount * 1000000000 wei == value);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the assert code block from the contract, which specified above.

Remediation Plan:

SOLVED: This finding was solved by removing the unnecessary assert block.

Commit ID: 4edda32d138d85e28ceb51e1519947078177a526

3.14 (HAL-14) IF CONDITIONS CAN BE OPTIMISED - INFORMATIONAL

Description:

Some **if** conditions on the contract using unnecessary boolean comparisons. Removing these boolean comparisons can optimize gas usage during the deployment of contract.

Code Location:

Listing 28: OperatorsManager.1.sol (Line 47)

```
46 modifier active(uint256 _index) {
47     if (Operators.getByIndex(_index).active == false) {
48         revert InactiveOperator(_index);
49     }
50     _;
51 }
```

Listing 29: OperatorsManager.1.sol (Line 61)

```
55 modifier operatorFeeRecipientOrAdmin(uint256 _index) {
56     if (msg.sender == LibOwnable._getAdmin()) {
57         _;
58         return;
59     }
60     Operators.Operator storage operator = Operators.getByIndex(
↳ _index);
61     if (operator.active == false) {
62         revert InactiveOperator(_index);
63     }
64     if (msg.sender != operator.feeRecipient) {
65         revert Errors.Unauthorized(msg.sender);
66     }
67     _;
68 }
```

Listing 30: OperatorsManager.1.sol (Line 78)

```
72 modifier operatorOrAdmin(uint256 _index) {
73     if (msg.sender == LibOwnable._getAdmin()) {
74         _;
75         return;
76     }
77     Operators.Operator storage operator = Operators.getByIndex(
↳ _index);
78     if (operator.active == false) {
79         revert InactiveOperator(_index);
80     }
81     if (msg.sender != operator.operator) {
82         revert Errors.Unauthorized(msg.sender);
83     }
84     _;
85 }
```

Listing 31: OperatorsManager.1.sol (Line 104)

```
 99 function addOperator(
100     string calldata _name,
101     address _operator,
102     address _feeRecipient
103 ) external onlyAdmin {
104     if (Operators.exists(_name) == true) {
105         revert OperatorAlreadyExists(_name);
106     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use the following pattern to consume less gas on contracts:

Listing 32: Example Fix

```
- if (variable == true) {
+ if (variable) {
    // function logic
}

- if (variable == false) {
+ if (!variable) {
    // function logic
}
```

Remediation Plan:

SOLVED: The issue was fixed by removing unnecessary boolean comparisons from their `if` conditions.

Commit ID: 4e946302e7485c663613b7067c12397dc6dfc357

3.15 (HAL-15) UNNECESSARY ADDITION OF TRANSFER AND DEPOSIT MASKS - INFORMATIONAL

Description:

The _onDeposit internal function on the RiverV1 contract uses TRANSFER_MASK and DEPOSIT_MASK variables to validate if user have permission to deposit ETH to the contract or transfer ETH to allowed addresses.

The DEPOSIT_MASK and TRANSFER_MASK variables are equal to 0x1 and 0x0 in order. The first onlyAllowed check tries to validate if the _depositor have permission for both deposit and transfer operations.

Listing 33: River.1.sol

```
(AllowlistAddress.get()).onlyAllowed(_depositor, DEPOSIT_MASK +
↳ TRANSFER_MASK);
```

However, addition of both these masks always returns 0x1 which is equals to DEPOSIT_MASK. As a result, use of this mathematical operation is unnecessary.

Code Location:

Listing 34: River.1.sol (Line 142)

```
function _onDeposit(
    address _depositor,
    address _recipient,
    uint256 _amount
) internal override {
    SharesManagerV1._mintShares(_depositor, _amount);
    if (_depositor == _recipient) {
        (AllowlistAddress.get()).onlyAllowed(_depositor,
↳ DEPOSIT_MASK); // this call reverts if unauthorized or denied
```

```
141     } else {
142         (AllowlistAddress.get()).onlyAllowed(_depositor,
 ↳ DEPOSIT_MASK + TRANSFER_MASK); // this call reverts if
 ↳ unauthorized or denied
143         (AllowlistAddress.get()).onlyAllowed(_recipient,
 ↳ TRANSFER_MASK);
144         _transfer(_depositor, _recipient, _amount);
145     }
146 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to correct the mathematical operation above. If it is intended behavior, remove the addition from onlyAllowed call.

Listing 35: Unnecessary use of addition operation

```
1 (AllowlistAddress.get()).onlyAllowed(_depositor, DEPOSIT_MASK +
 ↳ TRANSFER_MASK); // returns 0x1 + 0x0 = 0x1 which is DEPOSIT_MASK
```

Remediation Plan:

ACKNOWLEDGED: This finding was acknowledged.

> This is intended as it shows both rights are required to perform the action, but the fact that the mask is 0 means that it currently requires no special rights. This is a value that might be changed in the future depending on the protocol's evolution and future needs.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. 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:

```
WLSETHV1.mint(address,uint256) (contracts/src/WLSETH.1.sol#125-128) ignores return value by IRiverToken(RiverAddress.get()).transferFrom(msg.sender,address(this),_value) (contracts/src/WLSETH.1.sol#127)
WLSETHV1.burn(address,uint256) (contracts/src/WLSETH.1.sol#135-145) ignores return value by IRiverToken(RiverAddress.get()).transfer(_recipient,sharesAmount) (contracts/src/WLSETH.1.sol#144)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#9-11)
UnstructuredStorage.getStorageAddress(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#14-18) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#15-17)
UnstructuredStorage.getStorageBytes32(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#20-24) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#21-23)
UnstructuredStorage.getStorageUint256(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#26-30) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#27-29)
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#33-35)
UnstructuredStorage.setStorageAddress(bytes32,address) (contracts/src/libraries/UnstructuredStorage.sol#38-42) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#39-41)
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#45-47)
UnstructuredStorage.setStorageUint256(bytes32,uint256) (contracts/src/libraries/UnstructuredStorage.sol#50-54) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#51-53)
ApprovalsPerOwner.get(address,address) (contracts/src/state/shared/ApprovalsPerOwner.sol#12-22) uses assembly
        - INLINE ASM (contracts/src/state/shared/ApprovalsPerOwner.sol#17-19)
ApprovalsPerOwner.set(address,address,uint256) (contracts/src/state/shared/ApprovalsPerOwner.sol#24-38) uses assembly
        - INLINE ASM (contracts/src/state/shared/ApprovalsPerOwner.sol#33-35)
BalanceOf.get(address) (contracts/src/state/wlseth/BalanceOf.sol#11-21) uses assembly
        - INLINE ASM (contracts/src/state/wlseth/BalanceOf.sol#16-18)
BalanceOf.set(address,uint256) (contracts/src/state/wlseth/BalanceOf.sol#23-33) uses assembly
        - INLINE ASM (contracts/src/state/wlseth/BalanceOf.sol#28-30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) is never used and should be removed
UnstructuredStorage.getStorageBytes32(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#20-24) is never used and should be removed
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) is never used and should be removed
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.10 (contracts/src/Initializable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/WLSETH.1.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/interfaces/IRiverToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/libraries/UnstructuredStorage.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/shared/ApprovalsPerOwner.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/shared/RiverAddress.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/shared/Version.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/wlseth/BalanceOf.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
DepositManagerV1._depositValidator(bytes,bytes,bytes32) (contracts/src/components/DepositManager.1.sol#99-142) uses a dangerous strict equality:
        - require(bool,string)(address(this).balance == targetBalance,EXPECTING_DEPOSIT_TO_HAPPEN) (contracts/src/components/DepositManager.1.sol#140)
DepositManagerV1.depositToConsensusLayer(uint256) (contracts/src/components/DepositManager.1.sol#57-93) uses a dangerous strict equality:
        - validatorsToDeposit == 0 (contracts/src/components/DepositManager.1.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

DepositManagerV1._depositValidator(bytes,bytes,bytes32) (contracts/src/components/DepositManager.1.sol#99-142) has external calls inside a loop: DepositContractAddress.get().deposit{value: value}(_publicKey,abi.enc
odePacked(_withdrawalCredentials),_signature,depositDataRoot) (contracts/src/components/DepositManager.1.sol#134-139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in DepositManagerV1._depositValidator(bytes,bytes,bytes32) (contracts/src/components/DepositManager.1.sol#99-142):
        External calls:
        - DepositContractAddress.get().deposit{value: value}(_publicKey,abi.encodePacked(_withdrawalCredentials),_signature,depositDataRoot) (contracts/src/components/DepositManager.1.sol#134-139)
        Event emitted after the call(s):
        - FundedValidatorKey(_publicKey) (contracts/src/components/DepositManager.1.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

BytesLib.pad64(bytes) (contracts/src/libraries/BytesLib.sol#5-16) uses assembly
        - INLINE ASM (contracts/src/libraries/BytesLib.sol#10-12)
BytesLib.concat(bytes,bytes) (contracts/src/libraries/BytesLib.sol#18-90) uses assembly
        - INLINE ASM (contracts/src/libraries/BytesLib.sol#21-87)
BytesLib.slice(bytes,uint256,uint256) (contracts/src/libraries/BytesLib.sol#92-155) uses assembly
        - INLINE ASM (contracts/src/libraries/BytesLib.sol#102-152)
UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#9-11)
UnstructuredStorage.getStorageAddress(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#14-18) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#15-17)
UnstructuredStorage.getStorageBytes32(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#20-24) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#21-23)
UnstructuredStorage.getStorageUint256(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#26-30) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#27-29)
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#33-35)
UnstructuredStorage.setStorageAddress(bytes32,address) (contracts/src/libraries/UnstructuredStorage.sol#38-42) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#39-41)
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#45-47)
UnstructuredStorage.setStorageUint256(bytes32,uint256) (contracts/src/libraries/UnstructuredStorage.sol#50-54) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#51-53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

DepositContractAddress.set(IDepositContract) (contracts/src/state/river/DepositContractAddress.sol#15-17) is never used and should be removed
DepositManagerV1.initDepositManagerV1(address,bytes32) (contracts/src/components/DepositManager.1.sol#36-40) is never used and should be removed
UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) is never used and should be removed
UnstructuredStorage.setStorageAddress(bytes32,address) (contracts/src/libraries/UnstructuredStorage.sol#38-42) is never used and should be removed
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) is never used and should be removed
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) is never used and should be removed
WithdrawalCredentials.set(bytes32) (contracts/src/state/river/WithdrawalCredentials.sol#13-15) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#9-11)
UnstructuredStorage.getStorageAddress(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#14-18) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#15-17)
UnstructuredStorage.getStorageBytes32(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#20-24) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#21-23)
UnstructuredStorage.getStorageUint256(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#26-30) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#27-29)
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#33-35)
UnstructuredStorage.setStorageAddress(bytes32,address) (contracts/src/libraries/UnstructuredStorage.sol#38-42) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#39-41)
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#45-47)
UnstructuredStorage.setStorageUint256(bytes32,uint256) (contracts/src/libraries/UnstructuredStorage.sol#50-54) uses assembly
        - INLINE ASM (contracts/src/libraries/UnstructuredStorage.sol#51-53)
SharesPerOwner.get(address) (contracts/src/state/river/SharesPerOwner.sol#11-21) uses assembly
        - INLINE ASM (contracts/src/state/river/SharesPerOwner.sol#16-18)
SharesPerOwner.set(address,uint256) (contracts/src/state/river/SharesPerOwner.sol#23-33) uses assembly
        - INLINE ASM (contracts/src/state/river/SharesPerOwner.sol#28-30)
ApprovalsPerOwner.get(address,address) (contracts/src/state/shared/ApprovalsPerOwner.sol#12-22) uses assembly
        - INLINE ASM (contracts/src/state/shared/ApprovalsPerOwner.sol#17-19)
ApprovalsPerOwner.set(address,address,uint256) (contracts/src/state/shared/ApprovalsPerOwner.sol#24-38) uses assembly
        - INLINE ASM (contracts/src/state/shared/ApprovalsPerOwner.sol#33-35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Shares.set(uint256) (contracts/src/state/river/Shares.sol#13-15) is never used and should be removed
SharesManagerV1._mintRawShares(address,uint256) (contracts/src/components/SharesManager.1.sol#215-219) is never used and should be removed
SharesManagerV1._mintShares(address,uint256) (contracts/src/components/SharesManager.1.sol#200-210) is never used and should be removed
UnstructuredStorage.getStorageAddress(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#14-18) is never used and should be removed
UnstructuredStorage.getStorageBool(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#8-12) is never used and should be removed
UnstructuredStorage.getStorageBytes32(bytes32) (contracts/src/libraries/UnstructuredStorage.sol#20-24) is never used and should be removed
UnstructuredStorage.setStorageAddress(bytes32,address) (contracts/src/libraries/UnstructuredStorage.sol#38-42) is never used and should be removed
UnstructuredStorage.setStorageBool(bytes32,bool) (contracts/src/libraries/UnstructuredStorage.sol#32-36) is never used and should be removed
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (contracts/src/libraries/UnstructuredStorage.sol#44-48) is never used and should be removed
UnstructuredStorage.setStorageUint256(bytes32,uint256) (contracts/src/libraries/UnstructuredStorage.sol#50-54) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.10 (contracts/src/components/SharesManager.1.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/interfaces/IERC20.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/libraries/Errors.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/libraries/UnstructuredStorage.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/river/Shares.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/river/SharesPerOwner.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.10 (contracts/src/state/shared/ApprovalsPerOwner.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#64-73) ignores return value by Address.functionDelegateCall(newImplementation,data) (node_
modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#71)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#80-108) ignores return value by Address.functionDelegateCall(newImplementation,data)
 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#90)
ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#80-108) ignores return value by Address.functionDelegateCall(newImplementation,abi.e
ncodeWithSignature(upgradeTo(address),oldImplementation)) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#98-101)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#183-193) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).imp
lementation(),data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Modifier TransparentUpgradeableProxy.ifAdmin() (node_modules/@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#46-52) does not always execute _; or revertReference: https://github.com/crytic
/slither/wiki/Detector-Documentation#incorrect-modifier

Reentrancy in ERC1967Upgrade._upgradeToAndCallSecure(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#80-108):
        External calls:
        - Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#90)
        - Address.functionDelegateCall(newImplementation,abi.encodeWithSignature(upgradeTo(address),oldImplementation)) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#98-101)
        Event emitted after the call(s):
        - Upgraded(newImplementation) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#56)
                - _upgradeTo(newImplementation) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Proxy._delegate(address) (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#22-45) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#23-44)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#27-37) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33-35)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#196-216) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#208-211)
StorageSlot.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#52-56) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#53-55)
StorageSlot.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#61-65) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#62-64)
StorageSlot.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#70-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#71-73)
StorageSlot.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#79-83) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#80-82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

TUPProxy._beforeFallback() (contracts/src/TUPProxy.sol#43-49) compares to a boolean constant:
        -StorageSlot.getBooleanSlot(_PAUSE_SLOT).value == false || msg.sender == address(0) (contracts/src/TUPProxy.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

APPENDIX

5.1 Additional Notes

New PR (104):

PR#104 Commits

The Alluvial team made some changes after the audit finished. These changes were to the compound() method that sends protocol rewards to the treasury. Before the update, any user could call the compound() method. This method was removed from the contract.

Now, there are two methods to achieve that goal:

- pullELFees()
- sendELFees()

The new pullELFees() method can only be called via the River contract. Also, the internal _pullELFees() method has zero address checking and emits an event if **Oracle** calls the setBeaconData() method. From the security perspective, these changes have good defensive patterns overall.

New PR (107):

PR#107 Commits

The Alluvial team made another change to their River and SharesManager contracts. The purpose of this change was to correct the amount of mint. The contract used the shares instead of the amount of the underlying asset before this PR. Therefore, the minted token amount was incorrect for the depositor.

In the current situation, the SharesManager and River contracts use the underlying asset balance, and this solves the problem.

This small change has also been reviewed and confirmed by the Halborn team.

THANK YOU FOR CHOOSING

// HALBORN