// HALBORN

StaderLabs - ETHx

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 24th, 2023 - July 4th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/24/2023	Roberto Reigada
0.2	Document Updates	05/12/2023	Roberto Reigada
0.3	Draft Review	05/13/2023	Gokberk Gulgun
0.4	Draft Review	05/15/2023	Gabi Urrutia
1.0	Remediation Plan	05/30/2023	Roberto Reigada
1.1	Remediation Plan Review	05/30/2023	Gokberk Gulgun
1.2	Remediation Plan Review	06/01/2023	Gabi Urrutia
2.0	Document Updates	06/27/2023	Roberto Reigada
2.1	Document Updates	06/28/2023	Gokberk Gulgun
2.2	Document Updates Review	06/28/2023	Gokberk Gulgun
2.3	Document Updates Review	06/29/2023	Gabi Urrutia
3.0	Remediation Plan	06/30/2023	Roberto Reigada
3.1	Remediation Plan Review	07/04/2023	Gokberk Gulgun
3.2	Remediation Plan Review	07/04/2023	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

StaderLabs engaged Halborn to conduct a security audit on their smart contracts beginning on April 24th, 2023 and ending on May 12th, 2023. The security assessment was scoped to the smart contracts provided in the GitHub repository stader-labs/ethx.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned two full-time security engineers to audit the security of the smart contracts. The security engineers are blockchain and smart-contract security experts 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 StaderLabs team.

1.3 TEST APPROACH & METHODOLOGY

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

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

2. RISK METHODOLOGY

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

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

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

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

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

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

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

Attack Complexity (AX):

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

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 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) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 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

1. IN-SCOPE TREE & COMMIT :

The security assessment was scoped to the following smart contracts:

- Auction.sol
- ETHx.sol
- NodeELRewardVault.sol
- Penalty.sol
- PermissionedNodeRegistry.sol
- PermissionedPool.sol
- PermissionlessNodeRegistry.sol
- PermissionlessPool.sol
- PoolSelector.sol
- PoolUtils.sol
- SDCollateral.sol
- SocializingPool.sol
- StaderConfig.sol
- StaderInsuranceFund.sol
- StaderOracle.sol
- StaderStakePoolsManager.sol
- UserWithdrawalManager.sol
- ValidatorWithdrawalVault.sol
- VaultFactory.sol
- UtilLib.sol

Commit ID #1:

- eb9140b5b7779be3942e5bbcc8f52ebb33bf0df9

Commit ID #2:

- 4e63182f3dc6ab6572a2ab2c3cfc183f81f1507d

Commit ID #3:

- 21d516c32a0dd8742a97e4ec0689f08df706fa54

2. REMEDIATION & COMMIT :

Commit ID #1:

- 9c245db775127c29b18fa106145e5ccd68e7faa0

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	5	6

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
DENIAL OF SERVICE IN PERMISSIONLESSPOOL THROUGH A FORCED ETHER TRANSFER TO THE CONTRACT	Medium (5.5)	SOLVED - 05/30/2023
SLASHING OF A VALIDATOR CAN EXCEED THE ACTUAL PENALTY DUE	Medium (5.0)	PARTIALLY SOLVED - 06/30/2023
STADERORACLE CONTRACT CAN BE DOSED BY A MALICIOUS TRUSTED NODE	Medium (4.6)	SOLVED - 05/30/2023
WRONG CHECK IN USERWITHDRAWALMANAGER.REQUESTWITHDRAW FUNCTION	Low (3.7)	SOLVED - 05/30/2023
MISMANAGEMENT OF VALIDATOR STATUS LEADING TO POTENTIAL BLOCKING OF WITHDRAWALS	Low (3.7)	SOLVED - 05/30/2023
ABI.ENCODEPACKED() SHOULD NOT BE USED WITH DYNAMIC TYPES WHEN PASSING THE RESULT TO A HASH FUNCTION SUCH AS KECCAK256()	Low (2.3)	SOLVED - 05/30/2023
COMPUTEWITHDRAWVAULTADDRESS AND COMPUTENODEELREWARDVAULTADDRESS MAY RETURN INCORRECT ADDRESSES	Low (3.7)	SOLVED - 06/27/2023
VAULTPROXY IS SUBJECT TO STORAGE COLLISIONS	Low (3.7)	SOLVED - 06/30/2023
CHAINLINK'S LATESTROUNDDATA MIGHT RETURN STALE OR INCORRECT RESULTS	Informational (0.0)	ACKNOWLEDGED
FLOATING PRAGMA	Informational (0.0)	SOLVED - 05/30/2023
LACK OF ENFORCEMENT ON MINIMUM NUMBER OF TRUSTED NODES	Informational (0.0)	SOLVED - 05/30/2023
TYPO ON THE EVENT	Informational (0.0)	SOLVED - 05/30/2023
LOOP OPTIMIZATION	Informational (0.0)	SOLVED - 05/30/2023
MISSING/INCOMPLETE NATSPEC COMMENTS	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) DENIAL OF SERVICE IN PERMISSIONLESSPOOL THROUGH A FORCED ETHER TRANSFER TO THE CONTRACT - MEDIUM (5.5)

Description:

In the PermissionlessPool contract, the function stakeUserETHToBeaconChain() receives Ether from the pool manager and then deposits it on the beacon chain:

Listing 1: PermissionlessPool.sol (Line 155)

```
125 /**
126  * @notice receives eth from pool manager to deposit for
 ↳ validators on beacon chain
127  * @dev deposit validator taking care of pool capacity
128  */
129 function stakeUserETHToBeaconChain() external payable override
 ↳ nonReentrant {
130     UtilLib.onlyStaderContract(msg.sender, staderConfig,
 ↳ staderConfig.STAKE_POOL_MANAGER());
131     uint256 requiredValidators = msg.value / (staderConfig.
 ↳ getFullDepositSize() - DEPOSIT_NODE_BOND);
132     address nodeRegistryAddress = staderConfig.
 ↳ getPermissionlessNodeRegistry();
133     IPermissionlessNodeRegistry(nodeRegistryAddress).
 ↳ transferCollateralToPool(
134         requiredValidators * DEPOSIT_NODE_BOND
135     );
136
137     address vaultFactoryAddress = staderConfig.getVaultFactory();
138     address ethDepositContract = staderConfig.
 ↳ getETHDepositContract();
139     uint256 depositQueueStartIndex = IPermissionlessNodeRegistry(
 ↳ nodeRegistryAddress).nextQueuedValidatorIndex();
140     for (uint256 i = depositQueueStartIndex; i <
 ↳ requiredValidators + depositQueueStartIndex; i++) {
141         uint256 validatorId = IPermissionlessNodeRegistry(
 ↳ nodeRegistryAddress).queuedValidators(i);
```

```
142         fullDepositOnBeaconChain(
143             nodeRegistryAddress,
144             vaultFactoryAddress,
145             ethDepositContract,
146             validatorId,
147             staderConfig.getFullDepositSize()
148         );
149     }
150     IPermissionlessNodeRegistry(nodeRegistryAddress).
 ↳ updateNextQueuedValidatorIndex(
151         depositQueueStartIndex + requiredValidators
152     );
153     IPermissionlessNodeRegistry(nodeRegistryAddress).
 ↳ increaseTotalActiveValidatorCount(requiredValidators);
154     // balance must be 0 at this point
155     assert(address(this).balance == 0);
156 }
```

The assert statement is used to check that a certain condition is true, and if it is not, it will cause the contract to revert. In this case, the condition being checked is whether the balance of the contract is equal to 0, once the core logic of the stakeUserETHToBeaconChain() was executed. However, this asserts statement does not consider that the contract can receive Ether through, for example, a self-destruct transfer.

Based on this, a malicious user could:

1. Deploy the following contract:

Listing 2: Attack.sol (Lines 42,47)

```
1  contract Attack {
2      address PermissionlessPool;
3
4      constructor(address _permissionlessPool) {
5          PermissionlessPool = _permissionlessPool;
6      }
7
8      function attack() public payable {
9          address payable addr = payable(address(PermissionlessPool)
 ↳ );
10         selfdestruct(addr);
11     }
```

```
12 }
```

2. Call the attack() function passing to it just 1 Wei as msg.value.
3. The contract would self-destruct, forcing the transfer of 1 Wei to the PermissionlessPool contract.
4. Any future calls to stakeUserETHToBeaconChain() would revert to that 1 Wei would remain in the smart contract.

```
    │   │   │   └─ ← 0x1421dc38947cc1aab0ab2f26e877a453e92f408aa98d0ccf996585758d0a85f3
    │   │   ├─ [46723] DepositContract::deposit{value: 3100000000000000000}(0xa5dfd1c85461531c3bc9dc3fe21becd6f90f39bc550c
d0cb36, 0x90ad43c8aff4d8ef1579cae8c43b9832e417b8aff0d037775f0fcf803af48cbcb741d1a461aaaeec52cdb2a56be0746903d7240ff03d2c794
3e92f408aa98d0ccf996585758d0a85f3)
    │   │   │   ├─ emit DepositEvent(pubkey: 0xa5dfd1c85461531c3bc9dc3fe21becd6f90f39bc550cfef143a3d3b74b6f548961138576af46
amount: 0x0076be3707000000, signature: 0x90ad43c8aff4d8ef1579cae8c43b9832e417b8aff0d037775f0fcf803af48cbcb741d1a461aaaeec52
dex: 0x0100000000000000)
    │   │   │   ├─ [84] PRECOMPILE::keccak(a5dfd1c85461531c3bc9dc3fe21becd6f90f39bc550cfef143a3d3b74b6f548961138576af46ddc3
    │   │   │   │   └─ ← 0xf9ad0fb4a6fc67192ba7035504b3fa7c687a334004769232adadf02f7a2aee6f
    │   │   │   ├─ [84] PRECOMPILE::keccak(90ad43c8aff4d8ef1579cae8c43b9832e417b8aff0d037775f0fcf803af48cbcb741d1a461aaaeec
    │   │   │   │   └─ ← 0xbd8c3cacf5e07c1b0b213f98e06fe9bc4a54d50ca9dfc3de580a2a1be3897c61
    │   │   │   ├─ [84] PRECOMPILE::keccak(41f0bf4b1f6a1a0b9a5b151e817506c0ae4d7a28702d053bb3533defbde6721f0000000000000000
    │   │   │   │   └─ ← 0x8a3d248c7f0322e3cc035b2ce2826e783289045daf1193e43e5aa1dfa3a37b4d
    │   │   │   ├─ [84] PRECOMPILE::keccak(bd8c3cacf5e07c1b0b213f98e06fe9bc4a54d50ca9dfc3de580a2a1be3897c618a3d248c7f0322e3
    │   │   │   │   └─ ← 0xcc8bb8aa47e55a9fab3ad35ae90c326afc4c251b5652c3d89cba979fab005f16
    │   │   │   ├─ [84] PRECOMPILE::keccak(f9ad0fb4a6fc67192ba7035504b3fa7c687a334004769232adadf02f7a2aee6f0100000000000000
    │   │   │   │   └─ ← 0xa36ea7bcd5f65db6022b104c2cb130fea6dee30ea7f284c9b9ae5831a9d57664
    │   │   │   ├─ [84] PRECOMPILE::keccak(0076be3707000000000000000000000000000000000000000000000000000000cc8bb8aa47e55a9f
    │   │   │   │   └─ ← 0x8a51cfae97fe043d8d8212fab96a2eb9a2e974a48460fe80d0e185e2694b37ac
    │   │   │   ├─ [84] PRECOMPILE::keccak(a36ea7bcd5f65db6022b104c2cb130fea6dee30ea7f284c9b9ae5831a9d576648a51cfae97fe043d
    │   │   │   │   └─ ← 0x1421dc38947cc1aab0ab2f26e877a453e92f408aa98d0ccf996585758d0a85f3
    │   │   │   ├─ [84] PRECOMPILE::keccak(ef9d2f25c2802928cd31d7d83d243889c25148eabd28a58ad68a0d109579e7f11421dc38947cc1aa
    │   │   │   │   └─ ← 0x2e07957836b12d975b9f490dfb64048199ad9ab773b10a646f1fdd565ccc6224
    │   │   │   └─ ← ()
    │   │   ├─ [26733] PermissionlessNodeRegistry::updateDepositStatusAndBlock(1)
    │   │   │   ├─ [352] StaderConfig::PERMISSIONLESS_POOL() [staticcall]
    │   │   │   │   └─ ← 0x76d62e541b8d573110ca3eb9003e96426f530422a76712d1356f6c6ce50541ca
    │   │   │   ├─ [659] StaderConfig::onlyStaderContract(PermissionlessPool: [0x2cEe6B82e0ba79269CC14401b589c25a40198E12],
    │   │   │   │   └─ ← true
    │   │   │   ├─ emit UpdatedValidatorDepositBlock(validatorId: 1, depositBlock: 17132596)
    │   │   │   └─ ← ()
    │   │   ├─ emit ValidatorDepositedOnBeaconChain(validatorId: 1, pubKey: 0xa5dfd1c85461531c3bc9dc3fe21becd6f90f39bc550cf
    │   │   ├─ [23446] PermissionlessNodeRegistry::updateNextQueuedValidatorIndex(1)
    │   │   │   ├─ [352] StaderConfig::PERMISSIONLESS_POOL() [staticcall]
    │   │   │   │   └─ ← 0x76d62e541b8d573110ca3eb9003e96426f530422a76712d1356f6c6ce50541ca
    │   │   │   ├─ [659] StaderConfig::onlyStaderContract(PermissionlessPool: [0x2cEe6B82e0ba79269CC14401b589c25a40198E12],
    │   │   │   │   └─ ← true
    │   │   │   ├─ emit UpdatedNextQueuedValidatorIndex(nextQueuedValidatorIndex: 1)
    │   │   │   └─ ← ()
    │   │   ├─ [23697] PermissionlessNodeRegistry::increaseTotalActiveValidatorCount(1)
    │   │   │   ├─ [352] StaderConfig::PERMISSIONLESS_POOL() [staticcall]
    │   │   │   │   └─ ← 0x76d62e541b8d573110ca3eb9003e96426f530422a76712d1356f6c6ce50541ca
    │   │   │   ├─ [659] StaderConfig::onlyStaderContract(PermissionlessPool: [0x2cEe6B82e0ba79269CC14401b589c25a40198E12],
    │   │   │   │   └─ ← true
    │   │   │   ├─ emit IncreasedTotalActiveValidatorCount(totalActiveValidatorCount: 1)
    │   │   │   └─ ← ()
    │   │   └─ ← "Assertion violated"
    │   └─ ← "Assertion violated"
    └─ ← "Assertion violated"

Test result: FAILED. 0 passed; 1 failed; finished in 6.27s

Failing tests:
Encountered 1 failing test in test/StaderStakePoolsManager.t.sol:StaderStakePoolsManagerTests
[FAIL. Reason: Assertion violated] test_selfDestructToPermissionLessPool() (gas: 512690)
```

BVSS:

AO:A/AC:M/AX:L/C:N/I:H/A:C/D:M/Y:N/R:P/S:C (5.5)

Recommendation:

It is recommended to remove the `assert(address(this).balance == 0)` from the `stakeUserETHToBeaconChain()` function.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue by removing the assert condition in the following commit ID:

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.2 (HAL-02) SLASHING OF A VALIDATOR CAN EXCEED THE ACTUAL PENALTY DUE - MEDIUM (5.0)

Description:

The key problem is present in the settleFunds() function in the ValidatorWithdrawalVault contract, where the penaltyAmount is calculated and compared with operatorShare. If operatorShare is less than penaltyAmount, the function slashValidatorSD is called to make up for the deficit in SDCollateral:

Listing 3: ValidatorWithdrawalVault.sol (Lines 66-69)

```
54 function settleFunds() external override {
55     uint8 poolId = VaultProxy(payable(address(this))).poolId();
56     uint256 validatorId = VaultProxy(payable(address(this))).id();
57     IStaderConfig staderConfig = VaultProxy(payable(address(this))
↳ ).staderConfig();
58     address nodeRegistry = IPoolUtils(staderConfig.getPoolUtils())
↳ .getNodeRegistry(poolId);
59     if (msg.sender != nodeRegistry) {
60         revert CallerNotNodeRegistryContract();
61     }
62     (uint256 userSharePrelim, uint256 operatorShare, uint256
↳ protocolShare) = calculateValidatorWithdrawalShare();
63
64     uint256 penaltyAmount = getUpdatedPenaltyAmount(poolId,
↳ validatorId, staderConfig);
65
66     if (operatorShare < penaltyAmount) {
67         ISDCollateral(staderConfig.getSDCollateral()).
↳ slashValidatorSD(validatorId, poolId);
68         penaltyAmount = operatorShare;
69     }
70
71     uint256 userShare = userSharePrelim + penaltyAmount;
72     operatorShare = operatorShare - penaltyAmount;
73
74     // Final settlement
```

```
75      vaultSettleStatus = true;
76      IPenalty(staderConfig.getPenaltyContract()).
↳ markValidatorSettled(poolId, validatorId);
77      IStaderStakePoolManager(staderConfig.getStakePoolManager()).
↳ receiveWithdrawVaultUserShare{value: userShare}();
78      UtilLib.sendValue(payable(staderConfig.getStaderTreasury()),
↳ protocolShare);
79      IOperatorRewardsCollector(staderConfig.
↳ getOperatorRewardsCollector()).depositFor{value: operatorShare}(
80          getOperatorAddress(poolId, validatorId, staderConfig)
81      );
82      emit SettledFunds(userShare, operatorShare, protocolShare);
83  }
```

The main concern is the implementation of the slashValidatorSD() function‘:

Listing 4: SDCollateral.sol (Lines 80,89)

```
72  /// @notice slashes one validator equi. SD amount
73  /// @dev callable only by respective withdrawVaults
74  /// @param _validatorId validator SD collateral to slash
75  function slashValidatorSD(uint256 _validatorId, uint8 _poolId)
↳ external override nonReentrant {
76      address operator = UtilLib.getOperatorForValidSender(_poolId,
↳ _validatorId, msg.sender, staderConfig);
77      isPoolThresholdValid(_poolId);
78      PoolThresholdInfo storage poolThreshold =
↳ poolThresholdbyPoolId[_poolId];
79      uint256 sdToSlash = convertETHToSD(poolThreshold.minThreshold)
↳ ;
80      slashSD(operator, sdToSlash);
81  }
82
83  /// @notice used to slash operator SD, in case of operator default
84  /// @dev do provide SD approval to auction contract using `
↳ maxApproveSD()`
85  /// @param _operator which operator SD collateral to slash
86  /// @param _sdToSlash amount of SD to slash
87  function slashSD(address _operator, uint256 _sdToSlash) internal {
88      uint256 sdBalance = operatorSDBalance[_operator];
89      uint256 sdSlashed = Math.min(_sdToSlash, sdBalance);
90      if (sdSlashed == 0) {
```

```
91          return;
92      }
93      operatorSDBalance[_operator] -= sdSlashed;
94      IAuction(staderConfig.getAuctionContract()).createLot(
↳ sdSlashed);
95      emit SDSlashed(_operator, staderConfig.getAuctionContract(),
↳ sdSlashed);
96  }
```

It calculates sdToSlash as the minimum threshold of the pool (poolThreshold.minThreshold) rather than the actual deficit between operatorShare and penaltyAmount. The implementation of slashSD() then reduces the operator's SD balance by the minimum of sdToSlash and the operator's SD balance. This could result in slashing more SD tokens than required, in particular when the difference between operatorShare and penaltyAmount is smaller than poolThreshold.minThreshold.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to pass the actual deficit between operatorShare and penaltyAmount to the slashValidatorSD() function and compare it with poolThreshold.minThreshold using Math.min. This change would ensure that the amount of SD tokens slashed accurately reflects the actual penalty, protecting validators from unnecessary losses.

Remediation Plan:

PARTIALLY SOLVED: The team at Stader has recognized the concern highlighted. They confirm that this is an expected conduct of the system. In instances where the penalty surpasses operatorShare, an amount of SD equivalent to 0.4ETH is deducted. This procedure has been adequately communicated to all Node Operators.

It is also important to note that such a penalty does not have an impact on the funds staked by users.

4.3 (HAL-03) STADERORACLE CONTRACT CAN BE DOSED BY A MALICIOUS TRUSTED NODE - MEDIUM (4.6)

Description:

The StaderOracle contract implements the submitSDPrice() function:

Listing 5: StaderOracle.sol (Lines 244-255)

```
224 function submitSDPrice(SDPriceData calldata _sdPriceData) external
 ↳  override trustedNodeOnly {
225     if (_sdPriceData.reportingBlockNumber >= block.number) {
226         revert ReportingFutureBlockData();
227     }
228     if (_sdPriceData.reportingBlockNumber % updateFrequencyMap[
 ↳ SD_PRICE_UF] > 0) {
229         revert InvalidReportingBlock();
230     }
231     if (_sdPriceData.reportingBlockNumber <=
 ↳ lastReportedSDPriceData.reportingBlockNumber) {
232         revert ReportingPreviousCycleData();
233     }
234
235     // Get submission keys
236     bytes32 nodeSubmissionKey = keccak256(abi.encodePacked(msg.
 ↳ sender, _sdPriceData.reportingBlockNumber));
237     bytes32 submissionCountKey = keccak256(abi.encodePacked(
 ↳ _sdPriceData.reportingBlockNumber));
238     uint8 submissionCount = attestSubmission(nodeSubmissionKey,
 ↳ submissionCountKey);
239     insertSDPrice(_sdPriceData.sdPriceInETH);
240     // Emit SD Price submitted event
241     emit SDPriceSubmitted(msg.sender, _sdPriceData.sdPriceInETH,
 ↳ _sdPriceData.reportingBlockNumber, block.number);
242
243     // price can be derived once more than 66% percent oracles
 ↳ have submitted price
244     if ((submissionCount == (2 * trustedNodesCount) / 3 + 1)) {
245         lastReportedSDPriceData = _sdPriceData;
```

```
246         lastReportedSDPriceData.sdPriceInETH = getMedianValue(
 ↳ sdPrices);
247         uint256 len = sdPrices.length;
248         while (len > 0) {
249             sdPrices.pop();
250             len--;
251         }
252 
253         // Emit SD Price updated event
254         emit SDPriceUpdated(_sdPriceData.sdPriceInETH,
 ↳ _sdPriceData.reportingBlockNumber, block.number);
255     }
256 }
```

Once the 66% of the trusted nodes have submitted the price, the median value is calculated and then the total length of the sdPrices array is iterated in order to remove all its elements.

Firstly, this is highly inefficient, as the delete keyword could be used here instead.

Secondly, a malicious trusted node could perform the following exploit:

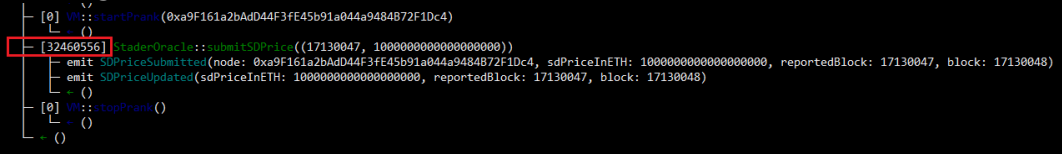
1. 2 trusted nodes are added to the StaderOracle contract by a manager.
2. The manager calls StaderOracle.setSDPriceUpdateFrequency() and sets it to 1.
3. TrustedNode1 which is a malicious-trusted node, calls 80000 times the submitSDPrice() function as shown in the code snippet below:

Listing 6: Malicious trusted node calls (Line 9)

```
1 vm.startPrank(trustedNode1);
2 SDPriceData memory _sdPriceData;
3 uint256 txAmount = 80000;
4 for(uint256 i; i < txAmount; ++i){
5     _sdPriceData = SDPriceData({
6         reportingBlockNumber: block.number - (txAmount - i),
7         sdPriceInETH: 1 ether
8     });
9     contract_StaderOracle.submitSDPrice(_sdPriceData);
```

```
10 }
11 vm.stopPrank();
```

4. TrustedNode2 now calls submitSDPrice() but the call reverts as the block gas limit is reached:



BVSS:

AO:A/AC:H/AX:L/C:N/I:N/A:C/D:M/Y:N/R:N/S:C (4.6)

Recommendation:

It is recommended to use delete to delete all the elements of an array in the StaderOracle.submitSDPrice() function. This would reduce the gas costs greatly and would make this attack described 3 times pricier. A malicious trusted node would require more than 150000 transactions in order to perform this exploit.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue by using the delete keyword to delete all the elements of the array. The issue was addressed in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.4 (HAL-04) WRONG CHECK IN USERWITHDRAWALMANAGER.REQUESTWITHDRAW FUNCTION - LOW (3.7)

Description:

The UserWithdrawalManager contract implements the requestWithdraw() function:

Listing 7: UserWithdrawalManager.sol (Lines 101,102,103,107)

```
 95 /**
 96  * @notice put a withdrawal request
 97  * @param _ethXAmount amount of ethX shares to withdraw
 98  * @param _owner owner of withdraw request to redeem
 99  */
100 function requestWithdraw(uint256 _ethXAmount, address _owner)
 ↳ external override whenNotPaused returns (uint256) {
101     if (_owner == address(0)) revert ZeroAddressReceived();
102     uint256 assets = IStaderStakePoolManager(staderConfig.
 ↳ getStakePoolManager()).previewWithdraw(_ethXAmount);
103     if (assets < staderConfig.getMinWithdrawAmount() || assets >
 ↳ staderConfig.getMaxWithdrawAmount()) {
104         revert InvalidWithdrawAmount();
105     }
106     if (requestIdsByUserAddress[msg.sender].length + 1 >
 ↳ maxNonRedeemedUserRequestCount) {
107         revert MaxLimitOnWithdrawRequestCountReached();
108     }
109     IERC20Upgradeable(staderConfig.getETHxToken()).
 ↳ safeTransferFrom(msg.sender, (address(this)), _ethXAmount);
110     ethRequestedForWithdraw += assets;
111     userWithdrawRequests[nextRequestId] = UserWithdrawInfo(payable
 ↳ (_owner), _ethXAmount, assets, 0, block.number);
112     requestIdsByUserAddress[_owner].push(nextRequestId);
113     emit WithdrawRequestReceived(msg.sender, _owner, nextRequestId
 ↳ , _ethXAmount, assets);
114     nextRequestId++;
115     return nextRequestId - 1;
116 }
```

This function checks that msg.sender does not have too many requests not redeemed yet. Although, msg.sender can request a withdrawal on behalf of other user; hence, this check should be done in the _owner parameter, not msg.sender as shown below:

Listing 8: UserWithdrawalManager.sol (Line 1)

```
1 if (requestIdsByUserAddress[_owner].length + 1 >
↳ maxNonRedeemedUserRequestCount) {
2     revert MaxLimitOnWithdrawRequestCountReached();
3 }
```

With the current implementation, a malicious user could perform multiple small requestWithdraw() calls in order to DOS the claims of a user:

Listing 9: UserWithdrawalManager.sol (Line 175)

```
162 function claim(uint256 _requestId) external override nonReentrant
 ↳ {
163     if (_requestId >= nextRequestIdToFinalize) {
164         revert requestIdNotFinalized(_requestId);
165     }
166     UserWithdrawInfo memory userRequest = userWithdrawRequests[
 ↳ _requestId];
167     if (msg.sender != userRequest.owner) {
168         revert CallerNotAuthorizedToRedeem();
169     }
170     // below is a default entry as no userRequest will be found
 ↳ for a redeemed request.
171     if (userRequest.ethExpected == 0) {
172         revert RequestAlreadyRedeemed(_requestId);
173     }
174     uint256 etherToTransfer = userRequest.ethFinalized;
175     deleteRequestId(_requestId, userRequest.owner);
176     sendValue(userRequest.owner, etherToTransfer);
177     emit RequestRedeemed(msg.sender, userRequest.owner,
 ↳ etherToTransfer);
178 }
```

Listing 10: UserWithdrawalManager.sol (Line 202)

```
198 function deleteRequestId(uint256 _requestId, address _owner)
 ↳ internal {
199     delete (userWithdrawRequests[_requestId]);
200     uint256 userRequestCount = requestIdsByUserAddress[_owner].
 ↳ length;
201     uint256[] storage requestIds = requestIdsByUserAddress[_owner
 ↳ ];
202     for (uint256 i = 0; i < userRequestCount; i++) {
203         if (_requestId == requestIds[i]) {
204             requestIds[i] = requestIds[userRequestCount - 1];
205             requestIds.pop();
206             return;
207         }
208     }
209     revert CannotFindRequestId();
210 }
```

Proof of Concept:

1. Bob performs, 80000 requestWithdraw() calls and sets Alice as the owner.
2. Alice performs a requestWithdraw() call, setting herself as the owner.
3. finalizeUserWithdrawalRequest() is called.
4. Alice tries to call claim() to claim her latest requestId, but the call reverts as the block gas limit is reached.

```
├─ [0] VM::startPrank(0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90)
│  └─ ← ()
├─ [33012882] UserWithdrawalManager::claim(80000)
│  ├─ [0] 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90::fallback{value: 100}()
│  │  └─ ← ()
│  ├─ emit RequestRedeemed(_sender: 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, _recipient: 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, _ethTransferred: 100)
│  └─ ← ()
├─ [0] VM::stopPrank()
│  └─ ← ()
└─ ← ()
```

BVSS:

AO:A/AC:H/AX:L/C:N/I:N/A:C/D:N/Y:M/R:N/S:U (3.7)

Recommendation:

It is recommended to update the UserWithdrawalManager.requestWithdraw() function as suggested.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.5 (HAL-05) MISMANAGEMENT OF VALIDATOR STATUS LEADING TO POTENTIAL BLOCKING OF WITHDRAWALS - LOW (3.7)

Description:

In the withdrawnValidators function, there is a potential risk when an oracle incorrectly flags a validator in the predeposit or initialized status as fully withdrawn. The issue arises from the system's inability to send <32 ETH to a withdrawn validator.

Code Location:

/PermissionedNodeRegistry.sol#L683-L689

Listing 11

```
    function isNonTerminalValidator(uint256 _validatorId) internal
↳ view returns (bool) {
        Validator memory validator = validatorRegistry[
↳ _validatorId];
        return
            !(validator.status == ValidatorStatus.WITHDRAWN ||
                validator.status == ValidatorStatus.FRONT_RUN ||
                validator.status == ValidatorStatus.
↳ INVALID_SIGNATURE);
    }
```

BVSS:

AO:A/AC:H/AX:L/C:N/I:N/A:C/D:N/Y:M/R:N/S:U (3.7)

Recommendation:

Consider using isActiveValidator instead of isNonTerminalValidator.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.6 (HAL-06) ABI.ENCODEPACKED() SHOULD NOT BE USED WITH DYNAMIC TYPES WHEN PASSING THE RESULT TO A HASH FUNCTION SUCH AS KECCAK256() - LOW (2.3)

Description:

Use abi.encode() instead, which will pad items to 32 bytes, which will prevent hash collisions (e.g. abi.encodePacked(0x123,0x456) => 0x123456 => abi.encodePacked(0x1,0x23456), but abi.encode(0x123,0x456)=> 0x0...1230...456). Unless there is a compelling reason, abi.encode should be preferred. If there is only one argument to abi.encodePacked() it can often be cast to bytes() or bytes32() instead.

Code Location:

StaderOracle.sol
- Line 110: abi.encodePacked(
- Line 119: abi.encodePacked(
- Line 179: abi.encodePacked(
- Line 190: abi.encodePacked(
- Line 236:
bytes32 nodeSubmissionKey = keccak256(abi.encodePacked(msg.sender, _sdPriceData.reportingBlockNumber));
- Line 237:
bytes32 submissionCountKey = keccak256(abi.encodePacked(_sdPriceData.reportingBlockNumber));
- Line 291: abi.encodePacked(
- Line 303: abi.encodePacked(
- Line 366: abi.encodePacked(
- Line 374: abi.encodePacked(
- Line 437:
abi.encodePacked(msg.sender, _mapd.index, _mapd.pageNumber,

encodedPubkeys)
- Line 439:
bytes32 submissionCountKey = keccak256(abi.encodePacked(_mapd.index,
_mapd.pageNumber, encodedPubkeys));

UtilLib.sol
- Line 134:
return sha256(abi.encodePacked(_pubkey, bytes16(0)));

SocializingPool.sol
- Line 168:
bytes32 node = keccak256(abi.encodePacked(_operator, _amountSD,
_amountETH));

PermissionedPool.sol
- Line 244:
bytes32 pubkey_root = sha256(abi.encodePacked(_pubkey, bytes16(0)));
- Line 246: abi.encodePacked(
- Line 247: sha256(abi.encodePacked(_signature[:64])),
- Line 248: sha256(abi.encodePacked(_signature[64:], bytes32(0)))
- Line 253: abi.encodePacked(
- Line 254:
sha256(abi.encodePacked(pubkey_root, _withdrawCredential)),
- Line 255:
sha256(abi.encodePacked(amount, bytes24(0), signature_root))

PermissionlessPool.sol
- Line 238:
bytes32 pubkey_root = sha256(abi.encodePacked(_pubkey, bytes16(0)));
- Line 240: abi.encodePacked(
- Line 241: sha256(abi.encodePacked(_signature[:64])),
- Line 242: sha256(abi.encodePacked(_signature[64:], bytes32(0)))
- Line 247: abi.encodePacked(
- Line 248:
sha256(abi.encodePacked(pubkey_root, _withdrawCredential)),
- Line 249:
sha256(abi.encodePacked(amount, bytes24(0), signature_root))

VaultFactory.sol

- Line 93:

```
return abi.encodePacked(bytes1(0x01), bytes11(0x0), address(
_withdrawVault));
```

BVSS:

AO:A/AC:M/AX:M/C:H/I:L/A:N/D:N/Y:N/R:P/S:C (2.3)

Recommendation:

Consider using abi.encode instead of abi.encodePacked.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.7 (HAL-07) COMPUTEWITHDRAWVAULTADDRESS AND COMPUTENODEELREWARDVAULTADDRESS MAY RETURN INCORRECT ADDRESSES - LOW (3.7)

Description:

When the vaultProxyImplementation address is updated by an admin, the computeWithdrawVaultAddress() and computeNodeELRewardVaultAddress() functions will return incorrect addresses as these functions use the ClonesUpgradeable.predictDeterministicAddress(vaultProxyImplementation, salt) library function to determine which address to return. Given the same vaultProxyImplementation and salt values, this function will always return the same address. However, the vaultProxyImplementation address can be changed by an admin call to updateVaultProxyAddress(), causing the returned addresses to point to non-existing contracts:

Listing 12: VaultFactory.sol (Line 95)

```
93 function updateVaultProxyAddress(address _vaultProxyImpl) external
↳  override onlyRole(DEFAULT_ADMIN_ROLE) {
94     UtilLib.checkNonZeroAddress(_vaultProxyImpl);
95     vaultProxyImplementation = _vaultProxyImpl;
96     emit UpdatedVaultProxyImplementation(vaultProxyImplementation)
↳ ;
97 }
```

The impact of this vulnerability can be quite severe. The PermissionlessPool.preDepositOnBeaconChain() function, which internally calls VaultFactory.computeWithdrawVaultAddress(), would deposit funds to the wrong address. This would effectively lock the funds in a non-recoverable manner as they are sent to non-existing contracts:

Listing 13: PermissionlessPool.sol (Line 95)

```
85 function preDepositOnBeaconChain(
86     bytes[] calldata _pubkey,
87     bytes[] calldata _preDepositSignature,
88     uint256 _operatorId,
89     uint256 _operatorTotalKeys
90 ) external payable nonReentrant {
91     UtilLib.onlyStaderContract(msg.sender, staderConfig,
 ↳ staderConfig.PERMISSIONLESS_NODE_REGISTRY());
92     address vaultFactory = staderConfig.getVaultFactory();
93     uint256 pubkeyCount = _pubkey.length;
94     for (uint256 i; i < pubkeyCount; ) {
95         address withdrawVault = IVaultFactory(vaultFactory).
 ↳ computeWithdrawVaultAddress(
96             INodeRegistry((staderConfig).
 ↳ getPermissionlessNodeRegistry()).POOL_ID(),
97             _operatorId,
98             _operatorTotalKeys + i
99         );
100         bytes memory withdrawCredential = IVaultFactory(
 ↳ vaultFactory).getValidatorWithdrawCredential(withdrawVault);
101 
102         bytes32 depositDataRoot = this.computeDepositDataRoot(
103             _pubkey[i],
104             _preDepositSignature[i],
105             withdrawCredential,
106             staderConfig.getPreDepositSize()
107         );
108         //slither-disable-next-line arbitrary-send-eth
109         IDepositContract(staderConfig.getETHDepositContract()).
 ↳ deposit{value: staderConfig.getPreDepositSize()}(
110             _pubkey[i],
111             withdrawCredential,
112             _preDepositSignature[i],
113             depositDataRoot
114         );
115         emit ValidatorPreDepositedOnBeaconChain(_pubkey[i]);
116         unchecked {
117             ++i;
118         }
119     }
120 }
```

BVSS:

AO:A/AC:H/AX:L/C:N/I:M/A:C/D:N/Y:N/R:N/S:U (3.7)

Recommendation:

It is recommended to prevent the admin from updating the vaultProxyImplementation. This will ensure that the functions computeWithdrawVaultAddress() and computeNodeELRewardVaultAddress() always return correct addresses.

Another option is creating a mapping to store the historical values of vaultProxyImplementation against the addresses returned by deployWithdrawVault() and deployNodeELRewardVault(). This way, even if the vaultProxyImplementation address is updated, the contract can reference the correct implementation address at the time of deployment, preventing it from pointing to non-existing contracts.

Remediation Plan:

SOLVED: The Stader team acknowledged the possible concern that might arise if updates to the ETHx smart contracts are not carefully executed in the future. To avoid any such complications, Stader's DAO will conduct a thorough examination of all future updates and will also seek multiple external audits. In the current contract setup, they've integrated several safety measures to avert any such issues. For instance, a feature like updateVaultProxyAddress is overseen by a time-delayed multi-signature mechanism, and they can pause the contracts when needed. These precautions are implemented to prevent potential issues from arising.

4.8 (HAL-08) VAULTPROXY IS SUBJECT TO STORAGE COLLISIONS - LOW (3.7)

Description:

The contract VaultProxy is a custom proxy implementation used by the StaderLabs team. This proxy will be used as the proxy to the ValidatorWithdrawalVault and NodeELRewardVault implementations.

VaultProxy storage

Name	Type	Slot	Offset	Bytes	Contract
vaultSettleStatus	bool	0	0	1	contracts/VaultProxy.sol:VaultProxy
isValidatorWithdrawalVault	bool	0	1	1	contracts/VaultProxy.sol:VaultProxy
isInitialized	bool	0	2	1	contracts/VaultProxy.sol:VaultProxy
poolId	uint8	0	3	1	contracts/VaultProxy.sol:VaultProxy
id	uint256	1	0	32	contracts/VaultProxy.sol:VaultProxy
owner	address	2	0	20	contracts/VaultProxy.sol:VaultProxy
staderConfig	contract IStaderConfig	3	0	20	contracts/VaultProxy.sol:VaultProxy

ValidatorWithdrawalVault storage

Name	Type	Slot	Offset	Bytes	Contract
vaultSettleStatus	bool	0	0	1	contracts/ValidatorWithdrawalVault.sol:ValidatorWithdrawalVault

NodeELRewardVault storage

Name	Type	Slot	Offset	Bytes	Contract

In the realm of smart contracts, and particularly in Solidity, it is essential to emphasize the consideration of storage handling when using proxy contracts.

Primarily, proxy contracts should ideally possess no storage of their own. The reason behind this stems from the fundamental purpose of a proxy contract, which is to delegate calls to an underlying logic contract, hence maintaining minimal functionality itself. This approach simplifies the upgradeability process, as changes to the logic contract do not necessitate modification to the storage layout of the proxy contract.

However, if a proxy contract does require its own storage, it is strongly recommended that the storage slots are positioned randomly or non-consecutively. This tactic mitigates the risk of collision with the

storage layout of the logic contract, thereby reducing the potential for critical issues.

Storage collision can occur when the proxy and logic contracts both attempt to access or modify the same storage slot. This can lead to unpredictable behavior, corrupt data, and in the worst-case scenario, make the contract vulnerable to exploits. The EVM does not differentiate storage spaces of different contracts in a delegatecall context. If the storage layouts are not carefully handled, writing to a storage location in the logic contract might unintentionally affect the state of the proxy contract, or vice versa.

With the current implementation of the ValidatorWithdrawalVault and NodeELRewardVault contracts, there is no issue as:

1. There is a storage collision in VaultProxy and ValidatorWithdrawalVault with the vaultSettleStatus state variable, although this is not a problem as both contracts save this state variable in exactly the same storage slot.
2. NodeELRewardVault uses no storage.

Although, if any of these contracts (ValidatorWithdrawalVault and NodeELRewardVault) were updated in the future, and they used new positions in their storage slots, a storage collision would occur. For this reason, the ideal practice is to strive for proxy contracts with no storage. If storage is unavoidable, it is crucial to ensure that the storage slots are organized randomly or non-consecutively to avoid potential storage collision.

BVSS:

AO:A/AC:H/AX:L/C:N/I:M/A:C/D:N/Y:N/R:N/S:U (3.7)

Recommendation:

It is recommended to strive for proxy contracts with no storage. If storage is unavoidable, it is crucial to ensure that the storage slots

are organized randomly or non-consecutively to avoid potential storage collision, ensuring a more robust and secure smart contract architecture.

Remediation Plan:

SOLVED: The Stader team has noted the highlighted concern and provided an explanation. The concern revolves around potential harm that could occur if updates are made to the ETHx smart contracts without proper consideration. To mitigate this risk, the Stader DAO will oversee any forthcoming updates and will obtain multiple audits to prevent such issues. It is important to note that under the present contract setup/configuration, this issue does not pose a concern.

4.9 (HAL-09) CHAINLINK'S LATESTROUNDDATA MIGHT RETURN STALE OR INCORRECT RESULTS - INFORMATIONAL (0.0)

Description:

The `getPORFeedData()` function in the `StaderOracle` contract fetches the price of the assets from Chainlink aggregators using the `latestRoundData()` function:

Listing 14: StaderOracle.sol (Lines 646-649)

```
637 function getPORFeedData()
638     internal
639     view
640     returns (
641         uint256,
642         uint256,
643         uint256
644     )
645 {
646     (, int256 totalETHBalanceInInt, , , ) = AggregatorV3Interface(
 ↳ staderConfig.getETHBalancePORFeedProxy())
647         .latestRoundData();
648     (, int256 totalETHXSupplyInInt, , , ) = AggregatorV3Interface(
 ↳ staderConfig.getETHXSupplyPORFeedProxy())
649         .latestRoundData();
650     return (uint256(totalETHBalanceInInt), uint256(
 ↳ totalETHXSupplyInInt), block.number);
651 }
```

However, there are no checks on `roundID` or `timeStamp`. If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandoning the oracle, chain congestion, vulnerability/attacks on the Chainlink system) consumers of this contract may continue using obsolete data.

On the other hand, according to Chainlink's documentation, latestRoundData() does not raise an error if no response has been reached, but returns 0, in this case feeding an incorrect price to the StaderOracle contract.

BVSS:

AO: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 add the following checks on the return data of the latestRoundData() function:

Listing 15: StaderOracle.sol (Lines 646-657)

```
637 function getPORFeedData()
638     internal
639     view
640     returns (
641         uint256,
642         uint256,
643         uint256
644     )
645 {
646     (uint80 baseRoundID, int256 totalETHBalanceInInt, , uint256
 ↳ baseTimestamp, uint80 baseAnsweredInRound) = AggregatorV3Interface
 ↳ (staderConfig.getETHBalancePORFeedProxy())
647         .latestRoundData();
648     require(totalETHBalanceInInt > 0, "ChainlinkPriceOracle:
 ↳ totalETHBalanceInInt <= 0");
649     require(baseAnsweredInRound >= baseRoundID, "
 ↳ ChainlinkPriceOracle: Stale price");
650     require(baseTimestamp > 0, "ChainlinkPriceOracle: Round not
 ↳ complete");
651     (baseRoundID, int256 totalETHXSupplyInInt, , baseTimestamp,
 ↳ baseAnsweredInRound) = AggregatorV3Interface(staderConfig.
 ↳ getETHXSupplyPORFeedProxy())
652         .latestRoundData();
653     require(totalETHXSupplyInInt > 0, "ChainlinkPriceOracle:
 ↳ totalETHXSupplyInInt <= 0");
```

```
654     require(baseAnsweredInRound >= baseRoundID, "
 ↳ ChainlinkPriceOracle: Stale price");
655     require(baseTimestamp > 0, "ChainlinkPriceOracle: Round not
 ↳ complete");
656     return (uint256(totalETHBalanceInInt), uint256(
 ↳ totalETHXSupplyInInt), block.number);
657 }
```

Remediation Plan:

ACKNOWLEDGED: The StaderLabs team acknowledged the concern raised with the following reason. Currently, the Chainlink implementation is not finalized. A toggle parameter currently prevents ER updates from Chainlink. When observing production contracts, it is evident that ER is currently fetched more from Oracles than Chainlink. The switch to Chainlink for ER feeds will only occur after thorough and extensive testing has been conducted.

4.10 (HAL-10) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Code Location:

Auction.sol
- Line 2: `pragma solidity ^0.8.16;`

ETHx.sol
- Line 2: `pragma solidity ^0.8.16;`

NodeELRewardVault.sol
- Line 2: `pragma solidity ^0.8.16;`

Penalty.sol
- Line 2: `pragma solidity ^0.8.16;`

PermissionedNodeRegistry.sol
- Line 2: `pragma solidity ^0.8.16;`

PermissionedPool.sol
- Line 2: `pragma solidity ^0.8.16;`

PermissionlessNodeRegistry.sol
- Line 2: `pragma solidity ^0.8.16;`

PermissionlessPool.sol
- Line 2: `pragma solidity ^0.8.16;`

PoolSelector.sol
- Line 2: pragma solidity ^0.8.16;

PoolUtils.sol
- Line 2: pragma solidity ^0.8.16;

SDCollateral.sol
- Line 2: pragma solidity ^0.8.16;

SocializingPool.sol
- Line 2: pragma solidity ^0.8.16;

StaderConfig.sol
- Line 2: pragma solidity ^0.8.16;

StaderInsuranceFund.sol
- Line 2: pragma solidity ^0.8.16;

StaderOracle.sol
- Line 2: pragma solidity ^0.8.16;

StaderStakePoolsManager.sol
- Line 2: pragma solidity ^0.8.16;

UserWithdrawalManager.sol
- Line 2: pragma solidity ^0.8.16;

ValidatorWithdrawalVault.sol
- Line 2: pragma solidity ^0.8.16;

VaultFactory.sol
- Line 2: pragma solidity ^0.8.16;

UtilLib.sol
- Line 2: pragma solidity ^0.8.16;

BVSS:

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

Recommendation:

Consider locking the pragma version in the smart contracts. It is not recommended to use a floating pragma in production.

For example: pragma solidity 0.8.16;

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.11 (HAL-11) LACK OF ENFORCEMENT ON MINIMUM NUMBER OF TRUSTED NODES - INFORMATIONAL (0.0)

Description:

The current implementation of the removeTrustedNode function allows the removal of trusted nodes without enforcing a minimum number of trusted nodes that should always be present in the system. As a result, it is theoretically possible to remove all trusted nodes, which could lead to the failure of any system processes that rely on these nodes.

If all trusted nodes are removed, it could potentially bring the system to a halt, disrupting services and leading to a loss of trust among users. Furthermore, it could potentially make the system more vulnerable to attacks.

Code Location:

StaderOracle.sol#LL84C14-L84C31

Listing 16

```
    function addTrustedNode(address _nodeAddress) external
↳ override {
        UtilLib.onlyManagerRole(msg.sender, staderConfig);
        UtilLib.checkNonZeroAddress(_nodeAddress);
        if (isTrustedNode[_nodeAddress]) {
            revert NodeAlreadyTrusted();
        }
        isTrustedNode[_nodeAddress] = true;
        trustedNodesCount++;

        emit TrustedNodeAdded(_nodeAddress);
    }

    /// @inheritdoc IStaderOracle
    function removeTrustedNode(address _nodeAddress) external
```

```
↳ override {
15          UtilLib.onlyManagerRole(msg.sender, staderConfig);
16          UtilLib.checkNonZeroAddress(_nodeAddress);
17          if (!isTrustedNode[_nodeAddress]) {
18              revert NodeNotTrusted();
19          }
20          isTrustedNode[_nodeAddress] = false;
21          trustedNodesCount--;
22
23          emit TrustedNodeRemoved(_nodeAddress);
24      }
```

BVSS:

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

Recommendation:

Add a check to ensure that there are always a minimum number of trusted nodes in the system.

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.12 (HAL-12) TYPO ON THE EVENT - INFORMATIONAL (0.0)

Description:

In the updateBidIncrement function, the event name BidInrementUpdated appears to have a typographical error. It seems like it should be BidIncrementUpdated instead of BidInrementUpdated. This can cause confusion and could lead to issues in event tracking or when using event logs for any kind of automation or tracking system.

Code Location:

Auction.sol#L153

Listing 17

```
    function updateBidIncrement(uint256 _bidIncrement) external override {
        UtilLib.onlyManagerRole(msg.sender, staderConfig);
        bidIncrement = _bidIncrement;
        emit BidInrementUpdated(_bidIncrement);
    }
```

BVSS:

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

Recommendation:

Typographical errors in code, especially in function or event names, should be fixed as soon as possible to prevent future issues. The corrected code should look like this:

Listing 18

```
function updateBidIncrement(uint256 _bidIncrement) external override {
    UtilLib.onlyManagerRole(msg.sender, staderConfig);
    bidIncrement = _bidIncrement;
    emit BidIncrementUpdated(_bidIncrement);  // Corrected event name
}
```

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.13 (HAL-13) LOOP OPTIMIZATION - INFORMATIONAL (0.0)

Description:

When a loop iterates many times, it causes the amount of gas required to execute the function to increase significantly. In Solidity, excessive looping can cause a function to use more than the maximum allowed gas, which causes the function to fail.

Code Location:

PermissionlessNodeRegistry.sol
- Line 135:
`for (uint256 i = 0; i < keyCount; i++){}`
- Line 190:
`for (uint256 i = 0; i < readyToDepositValidatorsLength; i++){}`
- Line 202:
`for (uint256 i = 0; i < frontRunValidatorsLength; i++){}`
- Line 209:
`for (uint256 i = 0; i < invalidSignatureValidatorsLength; i++)`
- Line 228:
`for (uint256 i = 0; i < withdrawnValidatorCount; i++){}`

BVSS:

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

Recommendation:

To reduce gas consumption, it is recommended to find ways to optimize the loop or potentially break the loop into smaller batches. The following pattern can also be used:

Listing 19

```
uint256 cachedLen = array.length;
for(uint i; i < cachedLen;){

  unchecked {
    ++i;
  }
}
```

Remediation Plan:

SOLVED: The StaderLabs team solved the issue in the following commit ID.

Commit ID : 9c245db775127c29b18fa106145e5ccd68e7faa0.

4.14 (HAL-14) MISSING/INCOMPLETE NATSPEC COMMENTS - INFORMATIONAL (0.0)

Description:

The functions are missing @param for some of their parameters. Given that **NatSpec** is an important part of code documentation, this affects code comprehension, auditability, and usability.

Code Location:

Contracts

BVSS:

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

Recommendation:

Consider adding in full **NatSpec** comments for all functions to have complete code documentation for future use.

Remediation Plan:

ACKNOWLEDGED: The StaderLabs team acknowledged this issue.

RECOMMENDATIONS OVERVIEW

1. Remove the `assert(address(this).balance == 0)` from the `stakeUserETHToBeaconChain()` function.
2. Use `delete` to delete all the elements of an array in the `StaderOracle.submitSDPrice()` function.
3. Update the `UserWithdrawalManager.requestWithdraw()` function as suggested.
4. Lock the pragma version in all the smart contracts.
5. Code adjustments should be made in the `deactivateNodeOperator` and `_deactivateNodeOperator` functions to not only flag the operator as inactive, but also to remove its permissions from the `permissionList`. This will prevent the operator from being re-onboarded or performing actions that require permissions after deactivation, thereby enhancing the system's security.
6. To mitigate the potential risk in the `withdrawnValidators` function, it is crucial to enhance the system's ability to handle cases when a validator is incorrectly flagged as fully withdrawn in the `predeposit` or `initialized` status by an oracle. This could be done by implementing a verification mechanism to double-check the status of the validator before executing the withdrawal function.
7. To address the identified issue in the `requestWithdraw` function, it is crucial to revise the implementation to prevent the `lastWithdrawReqTimestamp` from being updated on every call. Rather, it should be set only on the first call within the `withdrawDelay` period. This could be achieved by adding a condition that checks if the `withdrawDelay` period has elapsed since the last request before updating the `lastWithdrawReqTimestamp`.
8. To rectify the potential issue with the `slashSD` function, it is advised to implement a check ensuring that the `maxApproveSD` function has been executed before creating an auction. This could be integrated directly into the `slashSD` function or included in a separate pre-auction validation process. By ensuring the auction contract has the necessary approval to spend SD tokens prior to auction creation, you can prevent potential auction failures and ensure the smooth operation of the system.

AUTOMATED TESTING

6.1 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.

Slither results:

Auction.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/Auction.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/IAuction.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Auction.withdrawUnselectedBid(uint256) (contracts/Auction.sol#119-134):
        - (success) = address(msg.sender).call{value: withdrawalAmount}() (contracts/Auction.sol#130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

ETHx.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/ETHx.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

NodeELRewardVault.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.16 (contracts/NodeElRewardVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeElRewardVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in NodeElRewardVault.withdraw() (contracts/NodeElRewardVault.sol#50-76):
        - (success,None) = address(staderConfig.getStaderTreasury()).call{value: protocolShare}() (contracts/NodeElRewardVault.sol#63)
        - (success,None) = nodeRecipient.call{value: operatorShare}() (contracts/NodeElRewardVault.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

Penalty.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/Penalty.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPenalty.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IRatedV1.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

PermissionedNodeRegistry.sol

```
INFO:Detectors:
PermissionedNodeRegistry.getAllActiveValidators(uint256,uint256) (contracts/PermissionedNodeRegistry.sol#558-584) uses assembly
        - INLINE ASM (contracts/PermissionedNodeRegistry.sol#579-581)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
PermissionedNodeRegistry.addValidatorKeys(bytes[],bytes[],bytes[]) (contracts/PermissionedNodeRegistry.sol#136-174) has costly operations inside a loop:
        - nextValidatorId ++ (contracts/PermissionedNodeRegistry.sol#172)
PermissionedNodeRegistry.allocateValidatorsAndUpdateOperatorId(uint256) (contracts/PermissionedNodeRegistry.sol#185-233) has costly operations inside a loop:
        - operatorIdForExcessDeposit = 1_scope_0 (contracts/PermissionedNodeRegistry.sol#228)
PermissionedNodeRegistry._deactivateNodeOperator(uint256) (contracts/PermissionedNodeRegistry.sol#707-711) has costly operations inside a loop:
        - totalActiveOperatorCount -- (contracts/PermissionedNodeRegistry.sol#709)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/PermissionedNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionedNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionedPool.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IVaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/ISDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

PermissionedPool.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/PermissionedPool.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IDepositContract.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionedNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderInsuranceFund.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderPoolBase.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IVaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

PermissionlessNodeRegistry.sol

```
INFO:Detectors:
PermissionlessNodeRegistry.transferCollateralToPool(uint256) (contracts/PermissionlessNodeRegistry.sol#369-373) sends eth to arbitrary user
        Dangerous calls:
        - IPermissionlessPool(staderConfig.getPermissionlessPool()).receiveRemainingCollateralETH{value: _amount}() (contracts/PermissionlessNodeRegistry.sol#371)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in PermissionlessNodeRegistry.markValidatorReadyToDeposit(bytes[],bytes[],bytes[]) (contracts/PermissionlessNodeRegistry.sol#175-215):
        External calls:
        - IStaderInsuranceFund(staderConfig.getStaderInsuranceFund()).depositFund{value: frontRunValidatorsLength * FRONT_RUN_PENALTY}() (contracts/PermissionlessNodeRegistry.sol#198-200)
        State variables written after the call(s):
        - handleFrontRun(validatorId_scope_1) (contracts/PermissionlessNodeRegistry.sol#205)
                - validatorRegistry[_validatorId].status = ValidatorStatus.FRONT_RUN (contracts/PermissionlessNodeRegistry.sol#567)
        PermissionlessNodeRegistry.validatorRegistry (contracts/PermissionlessNodeRegistry.sol#44) can be used in cross function reentrancies:
        - PermissionlessNodeRegistry.getAllActiveValidators(uint256,uint256) (contracts/PermissionlessNodeRegistry.sol#464-490)
        - PermissionlessNodeRegistry.isActiveValidator(uint256) (contracts/PermissionlessNodeRegistry.sol#652-660)
        - PermissionlessNodeRegistry.isNonTerminalValidator(uint256) (contracts/PermissionlessNodeRegistry.sol#642-648)
        - PermissionlessNodeRegistry.markValidatorDeposited(uint256) (contracts/PermissionlessNodeRegistry.sol#674-676)
        - PermissionlessNodeRegistry.updateDepositStatusAndBlock(uint256) (contracts/PermissionlessNodeRegistry.sol#256-261)
        - PermissionlessNodeRegistry.validatorRegistry (contracts/PermissionlessNodeRegistry.sol#44)
        - PermissionlessNodeRegistry.withdrawnValidators(bytes[]) (contracts/PermissionlessNodeRegistry.sol#222-238)
```

```
Reentrancy in PermissionlessNodeRegistry.markValidatorReadyToDeposit(bytes[],bytes[],bytes[]) (contracts/PermissionlessNodeRegistry.sol#175-215):
        External calls:
        - IStaderInsuranceFund(staderConfig.getStaderInsuranceFund()).depositFund{value: frontRunValidatorsLength * FRONT_RUN_PENALTY}() (contracts/PermissionlessNodeRegistry.sol#198-200)
        - handleInvalidSignature(validatorId_scope_3) (contracts/PermissionlessNodeRegistry.sol#212)
            - (success) = address(_receiver).call{value: _amount}() (contracts/PermissionlessNodeRegistry.sol#624)
        State variables written after the call(s):
        - handleInvalidSignature(validatorId_scope_3) (contracts/PermissionlessNodeRegistry.sol#212)
            - validatorRegistry[_validatorId].status = ValidatorStatus.INVALID_SIGNATURE (contracts/PermissionlessNodeRegistry.sol#575)
        PermissionlessNodeRegistry.validatorRegistry (contracts/PermissionlessNodeRegistry.sol#44) can be used in cross function reentrancies:
        - PermissionlessNodeRegistry.getAllActiveValidators(uint256,uint256) (contracts/PermissionlessNodeRegistry.sol#464-490)
        - PermissionlessNodeRegistry.isActiveValidator(uint256) (contracts/PermissionlessNodeRegistry.sol#652-660)
        - PermissionlessNodeRegistry.isNonTerminalValidator(uint256) (contracts/PermissionlessNodeRegistry.sol#642-648)
        - PermissionlessNodeRegistry.markValidatorDeposited(uint256) (contracts/PermissionlessNodeRegistry.sol#674-676)
        - PermissionlessNodeRegistry.updateDepositStatusAndBlock(uint256) (contracts/PermissionlessNodeRegistry.sol#256-261)
        - PermissionlessNodeRegistry.validatorRegistry (contracts/PermissionlessNodeRegistry.sol#44)
        - PermissionlessNodeRegistry.withdrawnValidators(bytes[]) (contracts/PermissionlessNodeRegistry.sol#222-238)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
PermissionlessNodeRegistry.getAllActiveValidators(uint256,uint256) (contracts/PermissionlessNodeRegistry.sol#464-490) uses assembly
        - INLINE ASM (contracts/PermissionlessNodeRegistry.sol#485-487)
PermissionlessNodeRegistry.getAllSocializingPoolOptOutOperators(uint256,uint256) (contracts/PermissionlessNodeRegistry.sol#500-527) uses assembly
        - INLINE ASM (contracts/PermissionlessNodeRegistry.sol#522-524)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
PermissionlessNodeRegistry.addValidatorKeys(bytes[],bytes[],bytes[]) (contracts/PermissionlessNodeRegistry.sol#120-164) has costly operations inside a loop:
        - nextValidatorId ++ (contracts/PermissionlessNodeRegistry.sol#157)
PermissionlessNodeRegistry.markKeyReadyToDeposit(uint256) (contracts/PermissionlessNodeRegistry.sol#559-563) has costly operations inside a loop:
        - validatorQueueSize ++ (contracts/PermissionlessNodeRegistry.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/PermissionlessNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeElRewardVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionlessNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionlessPool.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderInsuranceFund.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IVaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/ISDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in PermissionlessNodeRegistry.sendValue(address,uint256) (contracts/PermissionlessNodeRegistry.sol#618-628):
        - (success) = address(_receiver).call{value: _amount}() (contracts/PermissionlessNodeRegistry.sol#624)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable PermissionlessNodeRegistry.markValidatorReadyToDeposit(bytes[],bytes[],bytes[]).validatorId_scope_1 (contracts/PermissionlessNodeRegistry.sol#203) is too similar to PermissionlessNodeRegistry.markValidatorReadyToDeposit(bytes[],
bytes[],bytes[]).validatorId_scope_3 (contracts/PermissionlessNodeRegistry.sol#210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

PermissionlessPool.sol

```
INFO:Detectors:
PermissionlessPool.fullDepositOnBeaconChain(address,address,address,uint256,uint256) (contracts/PermissionlessPool.sol#254-283) sends eth to arbitrary user
        Dangerous calls:
        - IDepositContract(_ethDepositContract).deposit{value: _DEPOSIT_SIZE}(pubkey,withdrawCredential,depositSignature,depositDataRoot) (contracts/PermissionlessPool.sol#275-280)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/PermissionlessPool.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IDepositContract.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionlessNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderPoolBase.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IVaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

PoolSelector.sol

```
INFO:Detectors:
PoolSelector.poolAllocationForExcessETHDeposit(uint256) (contracts/PoolSelector.sol#73-105) has costly operations inside a loop:
        - poolIdArrayIndexForExcessDeposit = i (contracts/PoolSelector.sol#101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/PoolSelector.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolSelector.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

PoolUtils.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.16 (contracts/PoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderPoolBase.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

SDCollateral.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/SDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/IAuction.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/ISDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

SocializingPool.sol

```
INFO:Detectors:
SocializingPool.handleRewards(RewardsData) (contracts/SocializingPool.sol#60-101) sends eth to arbitrary user
        Dangerous calls:
        - IStaderStakePoolManager(staderConfig.getStakePoolManager()).receiveExecutionLayerRewards{value: _rewardsData.userETHRewards}() (contracts/SocializingPool.sol#83-85)
        - (success) = address(staderConfig.getStaderTreasury()).call{value: _rewardsData.protocolETHRewards}() (contracts/SocializingPool.sol#87)
SocializingPool.claim(uint256[],uint256[],uint256[],bytes32[][]) (contracts/SocializingPool.sol#103-132) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = address(operatorRewardsAddr).call{value: totalAmountETH}() (contracts/SocializingPool.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/SocializingPool.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionlessNodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/ISocializingPool.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in SocializingPool.handleRewards(RewardsData) (contracts/SocializingPool.sol#60-101):
        - (success) = address(staderConfig.getStaderTreasury()).call{value: _rewardsData.protocolETHRewards}() (contracts/SocializingPool.sol#87)
Low level call in SocializingPool.claim(uint256[],uint256[],uint256[],bytes32[][]) (contracts/SocializingPool.sol#103-132):
        - (success,None) = address(operatorRewardsAddr).call{value: totalAmountETH}() (contracts/SocializingPool.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

StaderConfig.sol

```
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/StaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable StaderConfig.AUCTION_CONTRACT (contracts/StaderConfig.sol#45) is too similar to StaderConfig.updateAuctionContract(address)._auctionContract (contracts/StaderConfig.sol#197)
Variable StaderConfig.PENALTY_CONTRACT (contracts/StaderConfig.sol#46) is too similar to StaderConfig.updatePenaltyContract(address)._penaltyContract (contracts/StaderConfig.sol#205)
Variable StaderConfig.PERMISSIONED_POOL (contracts/StaderConfig.sol#47) is too similar to StaderConfig.updatePermissionedPool(address)._permissionedPool (contracts/StaderConfig.sol#209)
Variable StaderConfig.PERMISSIONLESS_POOL (contracts/StaderConfig.sol#50) is too similar to StaderConfig.updatePermissionlessPool(address)._permissionlessPool (contracts/StaderConfig.sol#217)
Variable StaderConfig.SD_COLLATERAL (contracts/StaderConfig.sol#42) is too similar to StaderConfig.updateSDCollateral(address)._sdCollateral (contracts/StaderConfig.sol#189)
Variable StaderConfig.STADER_ORACLE (contracts/StaderConfig.sol#44) is too similar to StaderConfig.updateStaderOracle(address)._staderOracle (contracts/StaderConfig.sol#201)
Variable StaderConfig.STADER_TREASURY (contracts/StaderConfig.sol#38) is too similar to StaderConfig.updateStaderTreasury(address)._staderTreasury (contracts/StaderConfig.sol#175)
Variable StaderConfig.VAULT_FACTORY (contracts/StaderConfig.sol#43) is too similar to StaderConfig.updateVaultFactory(address)._vaultFactory (contracts/StaderConfig.sol#193)
Variable StaderConfig.POOL_SELECTOR (contracts/StaderConfig.sol#41) is too similar to StaderConfig.updatePoolSelector(address)._poolSelector (contracts/StaderConfig.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

StaderInsuranceFund.sol

```
INFO:Detectors:
StaderInsuranceFund.reimburseUserFund(uint256) (contracts/StaderInsuranceFund.sol#60-66) sends eth to arbitrary user
        Dangerous calls:
        - IPermissionedPool(staderConfig.getPermissionedPool()).receiveInsuranceFund{value: _amount}() (contracts/StaderInsuranceFund.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.16 (contracts/StaderInsuranceFund.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPermissionedPool.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderInsuranceFund.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in StaderInsuranceFund.withdrawFund(uint256) (contracts/StaderInsuranceFund.sol#41-53):
        - (success) = address(msg.sender).call{value: _amount}() (contracts/StaderInsuranceFund.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

StaderOracle.sol

```
INFO:Detectors:
StaderOracle.submitSDPrice(SDPriceData) (contracts/StaderOracle.sol#224-256) has costly operations inside a loop:
        - sdPrices.pop() (contracts/StaderOracle.sol#249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/StaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/ISocializingPool.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable StaderOracle.attestSubmission(bytes32,bytes32)._nodeSubmissionKey (contracts/StaderOracle.sol#573) is too similar to StaderOracle.nodeSubmissionKeys (contracts/StaderOracle.sol#38)
Variable StaderOracle.attestSubmission(bytes32,bytes32)._submissionCountKey (contracts/StaderOracle.sol#573) is too similar to StaderOracle.submissionCountKeys (contracts/StaderOracle.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

StaderStakePoolsManager.sol

```
INFO:Detectors:
Reentrancy in StaderStakePoolsManager.depositETHOverTargetWeight() (contracts/StaderStakePoolsManager.sol#220-247):
        External calls:
        - (selectedPoolCapacity,poolIdArray) = IPoolSelector(staderConfig.getPoolSelector()).poolAllocationForExcessETHDeposit(availableETHForNewDeposit) (contracts/StaderStakePoolsManager.sol#227-229)
        - IStaderPoolBase(poolAddress).stakeUserETHToBeaconChain{value: validatorToDeposit * poolDepositSize}() (contracts/StaderStakePoolsManager.sol#244)
        External calls sending eth:
        - IStaderPoolBase(poolAddress).stakeUserETHToBeaconChain{value: validatorToDeposit * poolDepositSize}() (contracts/StaderStakePoolsManager.sol#244)
        State variables written after the call(s):
        - depositedPooledETH -= validatorToDeposit * poolDepositSize (contracts/StaderStakePoolsManager.sol#242)
        StaderStakePoolsManager.depositedPooledETH (contracts/StaderStakePoolsManager.sol#36) can be used in cross function reentrancies:
        - StaderStakePoolsManager._deposit(address,address,uint256,uint256) (contracts/StaderStakePoolsManager.sol#316-325)
        - StaderStakePoolsManager.depositedPooledETH (contracts/StaderStakePoolsManager.sol#36)
        - StaderStakePoolsManager.receiveEthFromAuction() (contracts/StaderStakePoolsManager.sol#83-86)
        - StaderStakePoolsManager.receiveExcessEthFromPool(uint8) (contracts/StaderStakePoolsManager.sol#92-95)
        - StaderStakePoolsManager.receiveExecutionLayerRewards() (contracts/StaderStakePoolsManager.sol#72-75)
        - StaderStakePoolsManager.receiveWithdrawVaultUserShare() (contracts/StaderStakePoolsManager.sol#78-81)
        - lastExcessETHDepositBlock = block.number (contracts/StaderStakePoolsManager.sol#241)
        StaderStakePoolsManager.lastExcessETHDepositBlock (contracts/StaderStakePoolsManager.sol#37) can be used in cross function reentrancies:
        - StaderStakePoolsManager.initialize(address,address) (contracts/StaderStakePoolsManager.sol#49-59)
        - StaderStakePoolsManager.lastExcessETHDepositBlock (contracts/StaderStakePoolsManager.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
StaderStakePoolsManager.depositETHOverTargetWeight() (contracts/StaderStakePoolsManager.sol#220-247) has costly operations inside a loop:
        - lastExcessETHDepositBlock = block.number (contracts/StaderStakePoolsManager.sol#241)
StaderStakePoolsManager.depositETHOverTargetWeight() (contracts/StaderStakePoolsManager.sol#220-247) has costly operations inside a loop:
        - depositedPooledETH -= validatorToDeposit * poolDepositSize (contracts/StaderStakePoolsManager.sol#242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/ETHx.sol#2) allows old versions
Pragma version^0.8.16 (contracts/StaderStakePoolsManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolSelector.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderPoolBase.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IUserWithdrawalManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in StaderStakePoolsManager.transferETHToUserWithdrawManager(uint256) (contracts/StaderStakePoolsManager.sol#102-111):


        - (success) = address(staderConfig.getUserWithdrawManager()).call{value: _amount}() (contracts/StaderStakePoolsManager.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

UserWithdrawalManager.sol

```
INFO:Detectors:
UserWithdrawalManager.finalizeUserWithdrawalRequest() (contracts/UserWithdrawalManager.sol#117-156) has costly operations inside a loop:
        - ethRequestedForWithdraw -= requiredEth (contracts/UserWithdrawalManager.sol#145)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Pragma version^0.8.16 (contracts/ETHx.sol#2) allows old versions
Pragma version^0.8.16 (contracts/UserWithdrawalManager.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderOracle.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IUserWithdrawalManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in UserWithdrawalManager.sendValue(address,uint256) (contracts/UserWithdrawalManager.sol#212-222):
        - (success) = _recipient.call{value: _amount}() (contracts/UserWithdrawalManager.sol#218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

ValidatorWithdrawalVault.sol

```
INFO:Detectors:
ValidatorWithdrawalVault.distributeRewards() (contracts/ValidatorWithdrawalVault.sol#57-78) sends eth to arbitrary user
        Dangerous calls:
        - IStaderStakePoolManager(staderConfig.getStakePoolManager()).receiveWithdrawVaultUserShare{value: userShare}() (contracts/ValidatorWithdrawalVault.sol#74)
ValidatorWithdrawalVault.settleFunds() (contracts/ValidatorWithdrawalVault.sol#88-103) sends eth to arbitrary user
        Dangerous calls:
        - IStaderStakePoolManager(staderConfig.getStakePoolManager()).receiveWithdrawVaultUserShare{value: userShare}() (contracts/ValidatorWithdrawalVault.sol#99)
ValidatorWithdrawalVault.sendValue(address,uint256) (contracts/ValidatorWithdrawalVault.sol#150-162) sends eth to arbitrary user
        Dangerous calls:
        - (success) = _recipient.call{value: _amount}() (contracts/ValidatorWithdrawalVault.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/ValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPenalty.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/ISDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in ValidatorWithdrawalVault.sendValue(address,uint256) (contracts/ValidatorWithdrawalVault.sol#150-162):
        - (success) = _recipient.call{value: _amount}() (contracts/ValidatorWithdrawalVault.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

VaultFactory.sol

```
INFO:Detectors:
ValidatorWithdrawalVault.distributeRewards() (contracts/ValidatorWithdrawalVault.sol#57-78) sends eth to arbitrary user
        Dangerous calls:
        - IStaderStakePoolManager(staderConfig.getStakePoolManager()).receiveWithdrawVaultUserShare{value: userShare}() (contracts/ValidatorWithdrawalVault.sol#74)
ValidatorWithdrawalVault.settleFunds() (contracts/ValidatorWithdrawalVault.sol#88-103) sends eth to arbitrary user
        Dangerous calls:
        - IStaderStakePoolManager(staderConfig.getStakePoolManager()).receiveWithdrawVaultUserShare{value: userShare}() (contracts/ValidatorWithdrawalVault.sol#99)
ValidatorWithdrawalVault.sendValue(address,uint256) (contracts/ValidatorWithdrawalVault.sol#150-162) sends eth to arbitrary user
        Dangerous calls:
        - (success) = _recipient.call{value: _amount}() (contracts/ValidatorWithdrawalVault.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/NodeELRewardVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/ValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/factory/VaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeELRewardVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPenalty.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderStakePoolManager.sol#3) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IVaultFactory.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/SDCollateral/ISDCollateral.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in NodeELRewardVault.withdraw() (contracts/NodeELRewardVault.sol#50-76):
        - (success,None) = address(staderConfig.getStaderTreasury()).call{value: protocolShare}() (contracts/NodeELRewardVault.sol#63)
        - (success,None) = nodeRecipient.call{value: operatorShare}() (contracts/NodeELRewardVault.sol#70)
Low level call in ValidatorWithdrawalVault.sendValue(address,uint256) (contracts/ValidatorWithdrawalVault.sol#150-162):
        - (success) = _recipient.call{value: _amount}() (contracts/ValidatorWithdrawalVault.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

UtilLib.sol

```
INFO:Detectors:
UtilLib.checkNonZeroAddress(address) (contracts/library/UtilLib.sol#20-22) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperator(uint8,address,IStaderConfig) (contracts/library/UtilLib.sol#113-121) is never used and should be removed
UtilLib.getNodeRecipientAddressByOperatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#104-111) is never used and should be removed
UtilLib.getNodeRecipientAddressByValidatorId(uint8,uint256,IStaderConfig) (contracts/library/UtilLib.sol#94-102) is never used and should be removed
UtilLib.getOperatorForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#64-79) is never used and should be removed
UtilLib.getPubkeyForValidSender(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#48-62) is never used and should be removed
UtilLib.getPubkeyRoot(bytes) (contracts/library/UtilLib.sol#128-135) is never used and should be removed
UtilLib.getValidatorSettleStatus(bytes,IStaderConfig) (contracts/library/UtilLib.sol#137-147) is never used and should be removed
UtilLib.onlyManagerRole(address,IStaderConfig) (contracts/library/UtilLib.sol#25-29) is never used and should be removed
UtilLib.onlyOperatorRole(address,IStaderConfig) (contracts/library/UtilLib.sol#31-35) is never used and should be removed
UtilLib.onlyStaderContract(address,IStaderConfig,bytes32) (contracts/library/UtilLib.sol#38-46) is never used and should be removed
UtilLib.onlyValidatorWithdrawVault(uint8,uint256,address,IStaderConfig) (contracts/library/UtilLib.sol#81-92) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.16 (contracts/interfaces/INodeRegistry.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IPoolUtils.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IStaderConfig.sol#2) allows old versions
Pragma version^0.8.16 (contracts/interfaces/IValidatorWithdrawalVault.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/UtilLib.sol#2) allows old versions
Pragma version^0.8.16 (contracts/library/ValidatorStatus.sol#2) allows old versions
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

- All the reentrancies flagged by Slither were checked individually

and are false positives.

- No major issues found by Slither.

6.2 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 to locate any vulnerabilities.

MythX results:

Auction.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
14	(SWC-123) Requirement Violation	Low	Requirement violation.
49	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
50	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
55	(SWC-123) Requirement Violation	Low	Requirement violation.
66	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
66	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
81	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
81	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
96	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
96	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
107	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
107	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
121	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
121	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

ETHx.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
17	(SWC-123) Requirement Violation	Low	Requirement violation.
18	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

NodeELRewardVault.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
14	(SWC-123) Requirement Violation	Low	Requirement violation.
38	(SWC-123) Requirement Violation	Low	Requirement violation.

Penalty.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
14	(SWC-123) Requirement Violation	Low	Requirement violation.
122	(SWC-123) Requirement Violation	Low	Requirement violation.

PermissionedNodeRegistry.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
20	(SWC-123) Requirement Violation	Low	Requirement violation.
111	(SWC-123) Requirement Violation	Low	Requirement violation.
307	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
358	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
360	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
605	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

PermissionedPool.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
56	(SWC-123) Requirement Violation	Low	Requirement violation.
160	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
160	(SWC-123) Requirement Violation	Low	Requirement violation.
167	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
192	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

PermissionlessNodeRegistry.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
21	(SWC-123) Requirement Violation	Low	Requirement violation.
234	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
246	(SWC-123) Requirement Violation	Low	Requirement violation.
258	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
260	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
275	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
288	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
289	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
552	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

PermissionlessPool.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
55	(SWC-123) Requirement Violation	Low	Requirement violation.
130	(SWC-123) Requirement Violation	Low	Requirement violation.
167	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
174	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
205	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

PoolSelector.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
14	(SWC-123) Requirement Violation	Low	Requirement violation.
56	(SWC-123) Requirement Violation	Low	Requirement violation.
57	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

PoolUtils.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

SDCollateral.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
16	(SWC-123) Requirement Violation	Low	Requirement violation.
243	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
244	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operator can overflow.
248	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
249	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operator can overflow.

SocializingPool.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
18	(SWC-123) Requirement Violation	Low	Requirement violation.
47	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
167	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
194	(SWC-123) Requirement Violation	Low	Requirement violation.

StaderConfig.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

StaderInsuranceFund.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
13	(SWC-123) Requirement Violation	Low	Requirement violation.
48	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
48	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
61	(SWC-123) Requirement Violation	Low	Requirement violation.

StaderOracle.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
15	(SWC-123) Requirement Violation	Low	Requirement violation.
98	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
167	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
205	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
220	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
225	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
241	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
254	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
282	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
356	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
415	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
449	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
461	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
558	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

StaderStakePoolsManager.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
55	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
63	(SWC-123) Requirement Violation	Low	Requirement violation.
103	(SWC-123) Requirement Violation	Low	Requirement violation.
142	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
221	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
221	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
241	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
273	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
296	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

UserWithdrawalManager.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
18	(SWC-123) Requirement Violation	Low	Requirement violation.
97	(SWC-123) Requirement Violation	Low	Requirement violation.
106	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
140	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

ValidatorWithdrawalVault.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
18	(SWC-123) Requirement Violation	Low	Requirement violation.
49	(SWC-123) Requirement Violation	Low	Requirement violation.

VaultFactory.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

UtilLib.sol

No output generated by MythX.

- The floating pragma was correctly flagged by MythX.
- block.number and block.hash are not used as a source of randomness.
- No major issues found by MythX.

THANK YOU FOR CHOOSING

// HALBORN