// Private Smart Contract Security Assessment

07.10.2024 - 07.19.2024

bnbX

Stader Labs

HALBORN

bnbX - Stader Labs

Prepared by: HALBORN

Last Updated 07/31/2024

Date of Engagement by: July 10th, 2024 - July 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	1	0	2	8

TABLE OF CONTENTS

1. Introduction

Stader Labs engaged Halborn to conduct a security assessment on their smart contracts beginning on **07-10-2024** and ending on **07-19-2024**. The security assessment was scoped to the smart contracts provided in the **https://github.com/stader-labs/bnbX** GitHub repository. Commit hashes and further details can be found in the Scope section of this report. The **bnbX** codebase in scope mainly consists of **liquid staking solution for BNB Smart Chain.**

2. Assessment Summary

Halborn was provided 5 days for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Stader Labs team. The main identified issues were:

- **Claim withdrawal may not available due to precision error.**
- **Incorrect logic prevents setting of Stader treasury address.**
- **Incorrect logic affects undelegation process.**

3. Test Approach And Methodology

`Halborn` performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Local testing with custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static analysis of security for scoped contract, and imported functions (`Slither`).

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/within the **remediation commit IDs**.
- Changes that occur outside of the scope of PRs.

4. 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.

4.1 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:

EXPLOITABILIY 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

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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:

$$E = \prod m_e$$

4.2 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

4.3 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.

METRICS:

SEVERITY 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: bnbX

(b) Assessed Commit ID: 2bba18c

(c) Items in scope:

- contracts/OperatorRegistry.sol
- contracts/StakeManagerV2.sol
- contracts/StakeManager.sol
- test/migration/Migration.t.sol
- script/foundry-scripts/migration/Migration.s.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- https:/https:/
- aee951caee951c

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	2	8

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CLAIM WITHDRAWAL MAY NOT BE AVAILABLE DUE TO PRECISION ERROR	HIGH	SOLVED - 07/24/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT LOGIC PREVENTS SETTING OF STADER TREASURY ADDRESS	LOW	SOLVED - 07/24/2024
INCORRECT LOGIC AFFECTS UNDELEGATION PROCESS	LOW	SOLVED - 07/24/2024
INCORRECT NATSPEC	INFORMATIONAL	SOLVED - 07/24/2024
POSSIBLE RE-ENTRANCY BEHAVIOR WHEN MIGRATING FUNDS	INFORMATIONAL	ACKNOWLEDGED
NON-REENTRANT MODIFIER ORDERING	INFORMATIONAL	SOLVED - 07/24/2024
OPEN TO-DOS	INFORMATIONAL	ACKNOWLEDGED
FLOATING PRAGMA	INFORMATIONAL	SOLVED - 07/24/2024
RISK OF EVM VERSION INCOMPATIBILITY ACROSS CHAINS	INFORMATIONAL	ACKNOWLEDGED
REDUNDANT VERIFICATION	INFORMATIONAL	SOLVED - 07/24/2024
USE OF MAGIC NUMBERS	INFORMATIONAL	SOLVED - 07/24/2024

7. FINDINGS & TECH DETAILS

7.1 CLAIM WITHDRAWAL MAY NOT BE AVAILABLE DUE TO PRECISION ERROR

// HIGH

Description

In the StakeManagerV2.sol contract, the claimWithdrawal() function is intended to facilitate the withdrawal of user funds. However, a precision error due to different calculations between contracts may prevent users from successfully withdrawing their funds.

```
function claimWithdrawal(uint256 _idx) external override whenNotPaused nonReentrant
returns (uint256) {
    WithdrawalRequest storage request = _extractRequest(msg.sender, _idx);
    if (request.claimed) revert AlreadyClaimed();

    BatchWithdrawalRequest memory batchRequest =
batchWithdrawalRequests[request.batchId];
    if (!batchRequest.isClaimable) revert Unbonding();

    request.claimed = true;
    uint256 amountInBnb = (batchRequest.amountInBnb * request.amountInBnbX) /
batchRequest.amountInBnbX;

    (bool success,) = payable(msg.sender).call{ value: amountInBnb }("");
    if (!success) revert TransferFailed();

    emit ClaimedWithdrawal(msg.sender, _idx, amountInBnb);
    return amountInBnb;
}
```

The issue stems from a dependency on external calculations using the StakeCredit contract address. Specifically, there is a discrepancy in share calculations between the startBatchUndelegation() and the StakeHub.undelegate() function, resulting in a mismatch of 1 wei in the bnbAmount calculation. This discrepancy between the two calculations will result in a failed execution of the claimWithdrawal() function after the completeBatchUndelegation() function has been called, as the funds claimed from the STAKE_HUB contract will be lower than the amount needed for withdrawal.

Proof of Concept

Take this scenario as an example, where a user makes a withdrawal request with the value 1000 :

```
function test_claimWithdrawal() public {
  vm.deal(user1, 100 ether);
  deal(address(bnbxAddrProxy), user1, 10 ether);
  vm.startPrank(user1);
  uint256 amountMinted = stakeManagerV2.delegate{value: 1 ether}("referral");
  amountMinted = 1000;

  /* --------------------------- withdrawal request --------------------------- */
  // there has to be a withdrawalrequest
  vm.startPrank(user1);
  IERC20(bnbxAddrProxy).approve(address(stakeManagerV2), type(uint256).max);
  stakeManagerV2.requestWithdraw(amountMinted);
  assertEq(stakeManagerV2.getWithdrawalRequestCount(), 1);
  assertEq(IERC20(bnbxAddrProxy).balanceOf(address(stakeManagerV2)), amountMinted);
  vm.stopPrank();

  /* ---------------------------- startBatch undelegation first ---------------------
------- */
  vm.startPrank(staderOperator);
  stakeManagerV2.startBatchUndelegation(1, address(0));
  uint batchWithdrawalRequestCount = stakeManagerV2.getBatchWithdrawalRequestCount();
  console.log("batchWithdrawalRequestCount: ", batchWithdrawalRequestCount);
  vm.stopPrank();

  /* ------------------- have to complete batch undelegation ------------------ */
  // this is to change the state of the batch request and make it claimable
  vm.startPrank(staderOperator);
  skip(7 days);
  uint firstUnbondingBatchIndexBefore = stakeManagerV2.firstUnbondingBatchIndex();

  /* ---------------------------------- deal ---------------------------------- */
  // if contract has that extra wei, execution succeeds
  // vm.deal(address(stakeManagerV2), 1); // UNCOMMENT THIS LINE TO SUCCESFULLY
EXECUTE
  /* ---------------------------------- deal ---------------------------------- */

  stakeManagerV2.completeBatchUndelegation();
  uint firstUnbondingBatchIndexAfter = stakeManagerV2.firstUnbondingBatchIndex();
  assertEq(firstUnbondingBatchIndexAfter, firstUnbondingBatchIndexBefore + 1);
  vm.stopPrank();

  vm.startPrank(user1);
  uint256 balanceOfUserBefore = user1.balance;
  vm.expectRevert(TransferFailed.selector);
  stakeManagerV2.claimWithdrawal(0);
}
```

1. An operator calls the `startBatchUndelegation()` function to store the withdrawal request.

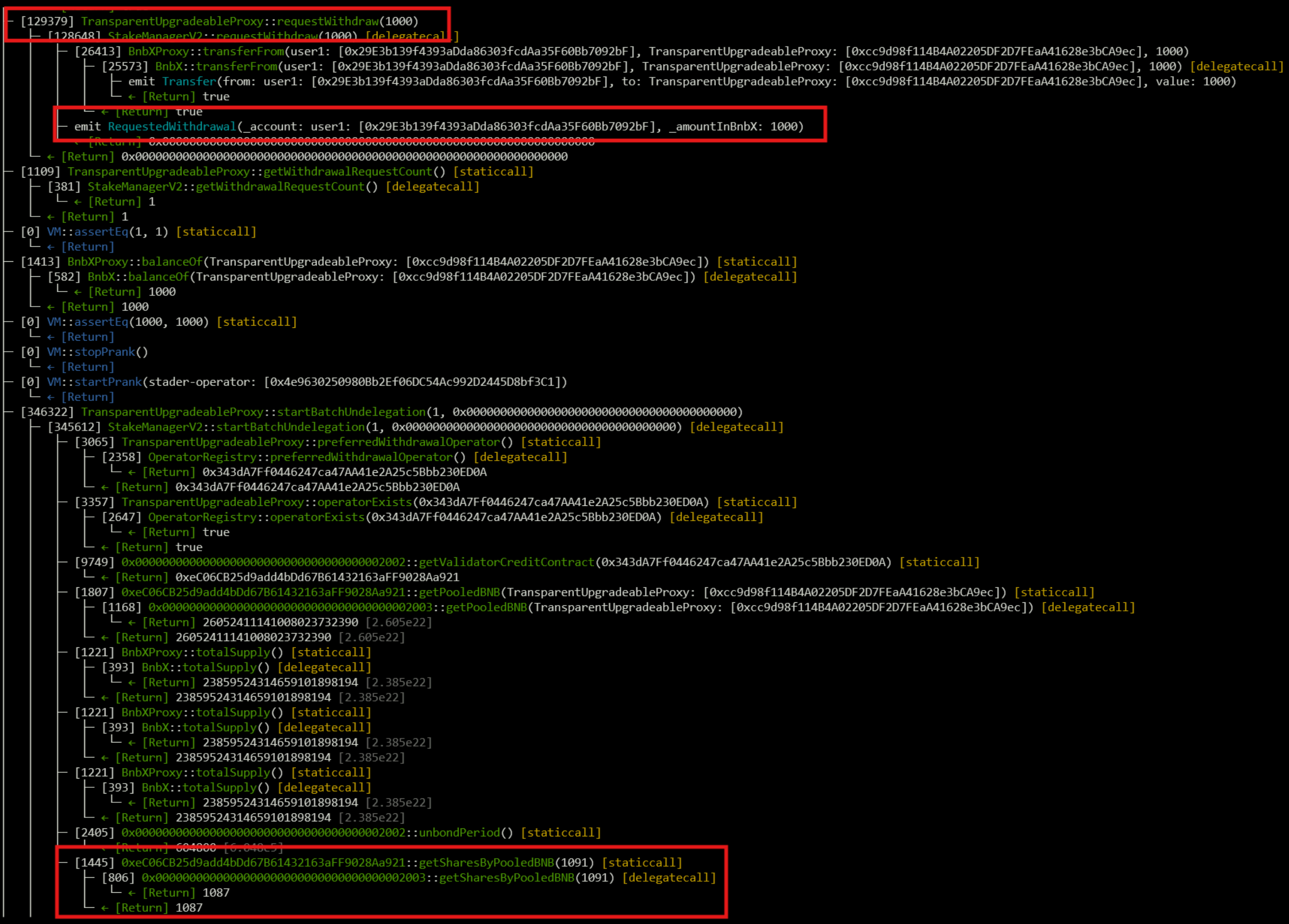
2. The `amountToWithdrawFromOperator` is calculated based on a call to the `convertBnbXToBnb(1000)` function, with an output of `1091`, storing this as the value to withdraw in the `batchWithdrawalRequests` array:

```
address creditContract = STAKE_HUB.getValidatorCreditContract(_operator);
uint256 amountInBnbXToBurn = _computeBnbXToBurn(_batchSize, creditContract)
uint256 amountToWithdrawFromOperator = convertBnbXToBnb(amountInBnbXToBurn)
totalDelegated -= amountToWithdrawFromOperator;

batchWithdrawalRequests.push(
  BatchWithdrawalRequest({
    amountInBnb: amountToWithdrawFromOperator,
    amountInBnbX: amountInBnbXToBurn,
    unlockTime: block.timestamp + STAKE_HUB.unbondPeriod(),
    operator: _operator,
    isClaimable: false
  })
);
```

3. This `1091` value is used to calculate the shares via the `StakeCredit` contract, which results in `1087` shares:

```
uint256 shares = IStakeCredit(creditContract).getSharesByPooledBNB(amountTo
```



```
├─ [129379] TransparentUpgradeableProxy::requestWithdraw(1000)
│   ├─ [128648] StakeManagerV2::requestWithdraw(1000) [delegatecall]
│   │   ├─ [26413] BnbXProxy::transferFrom(user1: [0x29E3b139f4393aDda86303fcdAa35F60Bb7092bF], TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1000)
│   │   │   ├─ [25573] BnbX::transferFrom(user1: [0x29E3b139f4393aDda86303fcdAa35F60Bb7092bF], TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1000) [delegatecall]
│   │   │   │   ├─ emit Transfer(from: user1: [0x29E3b139f4393aDda86303fcdAa35F60Bb7092bF], to: TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], value: 1000)
│   │   │   │   └─ ← [Return] true
│   │   │   └─ ← [Return] true
│   │   ├─ emit RequestedWithdrawal(_account: user1: [0x29E3b139f4393aDda86303fcdAa35F60Bb7092bF], _amountInBnbX: 1000)
│   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000000
│   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000000
├─ [1109] TransparentUpgradeableProxy::getWithdrawalRequestCount() [staticcall]
│   ├─ [381] StakeManagerV2::getWithdrawalRequestCount() [delegatecall]
│   │   └─ ← [Return] 1
│   └─ ← [Return] 1
├─ [0] VM::assertEq(1, 1) [staticcall]
│   └─ ← [Return]
├─ [1413] BnbXProxy::balanceOf(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [staticcall]
│   ├─ [582] BnbX::balanceOf(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [delegatecall]
│   │   └─ ← [Return] 1000
│   └─ ← [Return] 1000
├─ [0] VM::assertEq(1000, 1000) [staticcall]
│   └─ ← [Return]
├─ [0] VM::stopPrank()
│   └─ ← [Return]
├─ [0] VM::startPrank(stader-operator: [0x4e9630250980Bb2Ef06DC54Ac992D2445D8bf3C1])
│   └─ ← [Return]
├─ [346322] TransparentUpgradeableProxy::startBatchUndelegation(1, 0x0000000000000000000000000000000000000000)
│   ├─ [345612] StakeManagerV2::startBatchUndelegation(1, 0x0000000000000000000000000000000000000000) [delegatecall]
│   │   ├─ [3065] TransparentUpgradeableProxy::preferredWithdrawalOperator() [staticcall]
│   │   │   ├─ [2358] OperatorRegistry::preferredWithdrawalOperator() [delegatecall]
│   │   │   │   └─ ← [Return] 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A
│   │   │   └─ ← [Return] 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A
│   │   ├─ [3357] TransparentUpgradeableProxy::operatorExists(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [staticcall]
│   │   │   ├─ [2647] OperatorRegistry::operatorExists(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [delegatecall]
│   │   │   │   └─ ← [Return] true
│   │   │   └─ ← [Return] true
│   │   ├─ [9749] 0x0000000000000000000000000000000000002002::getValidatorCreditContract(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [staticcall]
│   │   │   └─ ← [Return] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921
│   │   ├─ [1807] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [staticcall]
│   │   │   ├─ [1168] 0x0000000000000000000000000000000000002003::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [delegatecall]
│   │   │   │   └─ ← [Return] 26052411141008023732390 [2.605e22]
│   │   │   └─ ← [Return] 26052411141008023732390 [2.605e22]
│   │   ├─ [1221] BnbXProxy::totalSupply() [staticcall]
│   │   │   ├─ [393] BnbX::totalSupply() [delegatecall]
│   │   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   ├─ [1221] BnbXProxy::totalSupply() [staticcall]
│   │   │   ├─ [393] BnbX::totalSupply() [delegatecall]
│   │   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   ├─ [1221] BnbXProxy::totalSupply() [staticcall]
│   │   │   ├─ [393] BnbX::totalSupply() [delegatecall]
│   │   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   ├─ [2405] 0x0000000000000000000000000000000000002002::unbondPeriod() [staticcall]
│   │   │   └─ ← [Return] 604800 [6.048e5]
│   │   ├─ [1445] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::getSharesByPooledBNB(1091) [staticcall]
│   │   │   ├─ [806] 0x0000000000000000000000000000000000002003::getSharesByPooledBNB(1091) [delegatecall]
│   │   │   │   └─ ← [Return] 1087
│   │   │   └─ ← [Return] 1087
```

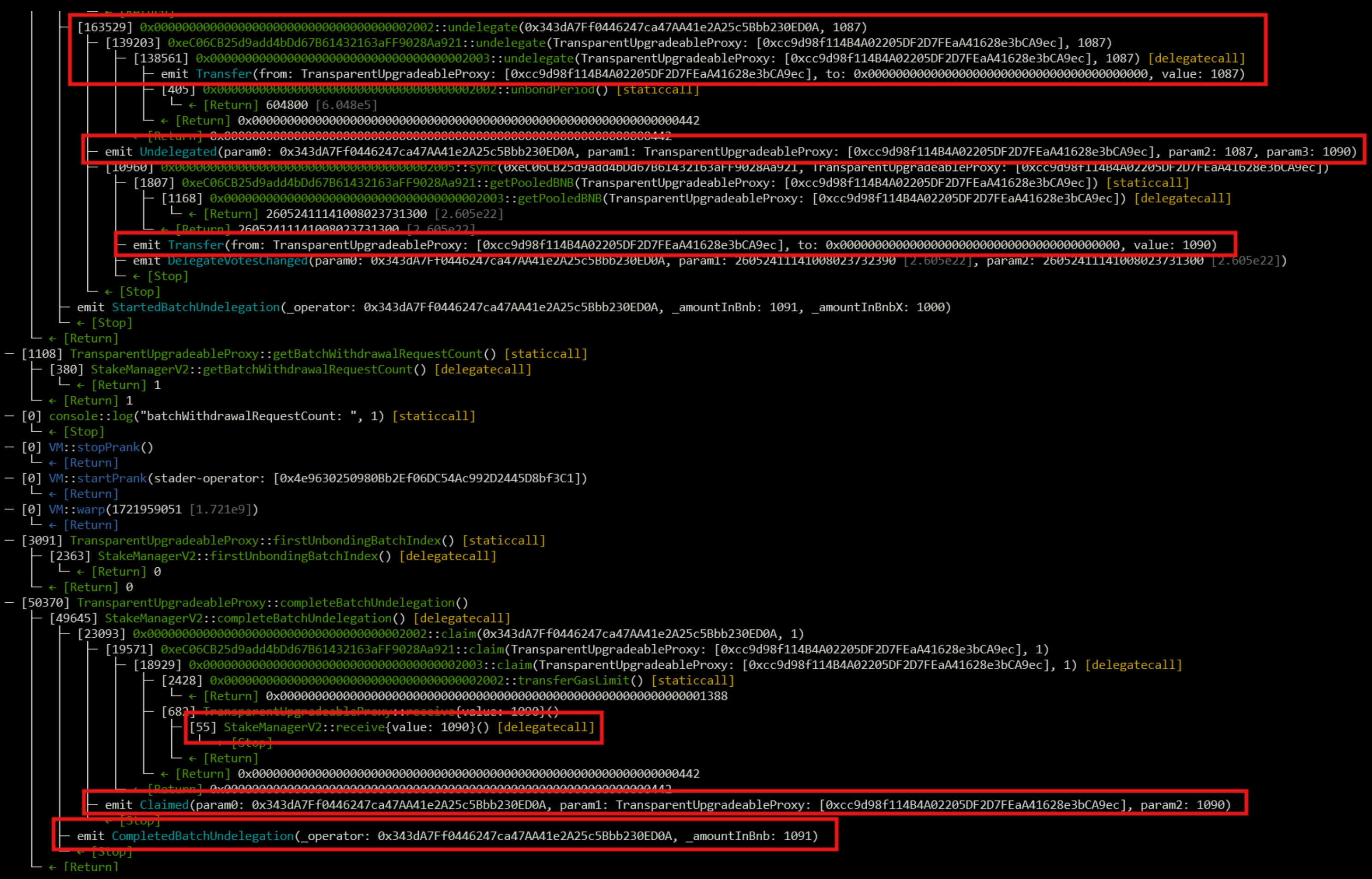
4. That value of 1087 shares is passed to the `STAKE_HUB.undelegate()` function:

```
STAKE_HUB.undelegate(_operator, shares);
```

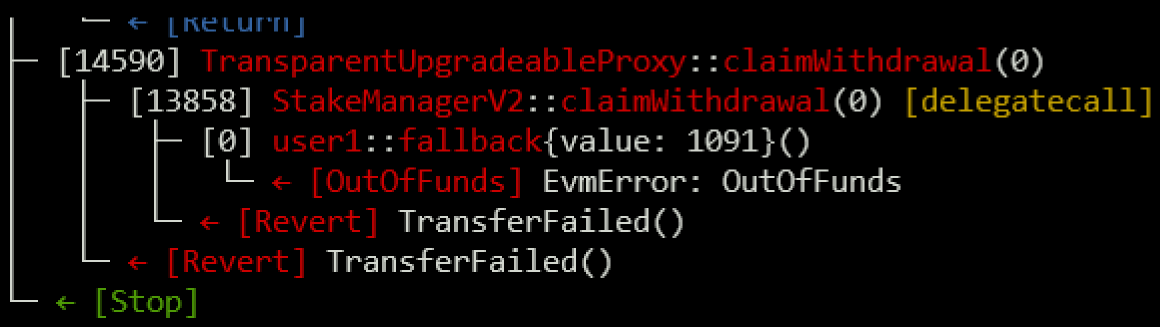
5. The `STAKE_HUB.undelegate()` function calculates the `bnbAmount` by a call to the Stake Credit contract:

```
uint256 bnbAmount = IStakeCredit(valInfo.creditContract).undelegate(delegat
```

6. The `bnbAmount` returned from the `STAKE_HUB.undelegate()` function is `1090` instead of the expected `1091`.



```
    │   │   ├─ [163529] 0x0000000000000000000000000000000000002002::undelegate(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, 1087)
    │   │   │   ├─ [139203] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::undelegate(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1087)
    │   │   │   │   ├─ [138561] 0x0000000000000000000000000000000000002003::undelegate(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1087) [delegatecall]
    │   │   │   │   │   ├─ emit Transfer(from: TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], to: 0x0000000000000000000000000000000000000000, value: 1087)
    │   │   │   │   │   ├─ [405] 0x0000000000000000000000000000000000002002::unbondPeriod() [staticcall]
    │   │   │   │   │   │   └─ ← [Return] 604800 [6.048e5]
    │   │   │   │   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000442
    │   │   │   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000442
    │   │   │   ├─ emit Undelegated(param0: 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, param1: TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], param2: 1087, param3: 1090)
    │   │   │   ├─ [10960] 0x0000000000000000000000000000000000002005::sync(0xeC06CB25d9add4bDd67B61432163aFF9028Aa921, TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec])
    │   │   │   │   ├─ [1807] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [staticcall]
    │   │   │   │   │   ├─ [1168] 0x0000000000000000000000000000000000002003::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [delegatecall]
    │   │   │   │   │   │   └─ ← [Return] 26052411141008023731300 [2.605e22]
    │   │   │   │   │   └─ ← [Return] 26052411141008023731300 [2.605e22]
    │   │   │   │   ├─ emit Transfer(from: TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], to: 0x0000000000000000000000000000000000000000, value: 1090)
    │   │   │   │   ├─ emit DelegateVotesChanged(param0: 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, param1: 26052411141008023732390 [2.605e22], param2: 26052411141008023731300 [2.605e22])
    │   │   │   │   └─ ← [Stop]
    │   │   │   └─ ← [Stop]
    │   │   ├─ emit StartedBatchUndelegation(_operator: 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, _amountInBnb: 1091, _amountInBnbX: 1000)
    │   │   └─ ← [Stop]
    │   └─ ← [Return]
    ├─ [1108] TransparentUpgradeableProxy::getBatchWithdrawalRequestCount() [staticcall]
    │   ├─ [380] StakeManagerV2::getBatchWithdrawalRequestCount() [delegatecall]
    │   │   └─ ← [Return] 1
    │   └─ ← [Return] 1
    ├─ [0] console::log("batchWithdrawalRequestCount: ", 1) [staticcall]
    │   └─ ← [Stop]
    ├─ [0] VM::stopPrank()
    │   └─ ← [Return]
    ├─ [0] VM::startPrank(stader-operator: [0x4e9630250980Bb2Ef06DC54Ac992D2445D8bf3C1])
    │   └─ ← [Return]
    ├─ [0] VM::warp(1721959051 [1.721e9])
    │   └─ ← [Return]
    ├─ [3091] TransparentUpgradeableProxy::firstUnbondingBatchIndex() [staticcall]
    │   ├─ [2363] StakeManagerV2::firstUnbondingBatchIndex() [delegatecall]
    │   │   └─ ← [Return] 0
    │   └─ ← [Return] 0
    ├─ [50370] TransparentUpgradeableProxy::completeBatchUndelegation()
    │   ├─ [49645] StakeManagerV2::completeBatchUndelegation() [delegatecall]
    │   │   ├─ [23093] 0x0000000000000000000000000000000000002002::claim(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, 1)
    │   │   │   ├─ [19571] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::claim(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1)
    │   │   │   │   ├─ [18929] 0x0000000000000000000000000000000000002003::claim(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], 1) [delegatecall]
    │   │   │   │   │   ├─ [2428] 0x0000000000000000000000000000000000002002::transferGasLimit() [staticcall]
    │   │   │   │   │   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000001388
    │   │   │   │   │   ├─ [682] TransparentUpgradeableProxy::receive{value: 1090}()
    │   │   │   │   │   │   ├─ [55] StakeManagerV2::receive{value: 1090}() [delegatecall]
    │   │   │   │   │   │   │   └─ ← [Stop]
    │   │   │   │   │   │   └─ ← [Return]
    │   │   │   │   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000442
    │   │   │   │   └─ ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000442
    │   │   │   ├─ emit Claimed(param0: 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, param1: TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec], param2: 1090)
    │   │   │   └─ ← [Stop]
    │   │   ├─ emit CompletedBatchUndelegation(_operator: 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A, _amountInBnb: 1091)
    │   │   └─ ← [Stop]
    │   └─ ← [Return]
```



```
    │   └─ ← [Return]
    ├─ [14590] TransparentUpgradeableProxy::claimWithdrawal(0)
    │   ├─ [13858] StakeManagerV2::claimWithdrawal(0) [delegatecall]
    │   │   ├─ [0] user1::fallback{value: 1091}()
    │   │   │   └─ ← [OutOfFunds] EvmError: OutOfFunds
    │   │   └─ ← [Revert] TransferFailed()
    │   └─ ← [Revert] TransferFailed()
    └─ ← [Stop]
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:H/Y:N (7.5)

Recommendation

Ensure that the claimed values from the external call to the `STAKE_HUB` contract when executing `completeBatchUndelegation()` match the expected withdrawal values calculated in the `amountInBnb` from the `batchWithdrawalRequest` stored so that the native balance of the `StakeManagerV2` contract is sufficient for users calling the `claimWithdrawal()` function.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by utilizing the `IStakeCredit.getPooledBNBByShares()` from the logic to calculate the amount of BnB to return to users.

Remediation Hash

https://github.com/stader-labs/bnbX/blob/aee951c5477fa8091f25386980fddeac905f9e20/contracts/StakeManagerV2.sol#L196

References

StakeManagerV2.sol#L130
StakeManagerV2.sol#L150
StakeManagerV2.sol#L192

7.2 INCORRECT LOGIC PREVENTS SETTING OF STADER TREASURY ADDRESS

// LOW

Description

In the `StakeManagerV2.sol` contract, the `setStaderTreasury()` function is used to set the address of the Stader treasury. However, the function is not implemented correctly, as it reverts if the `_staderTreasury` address is not the zero address:

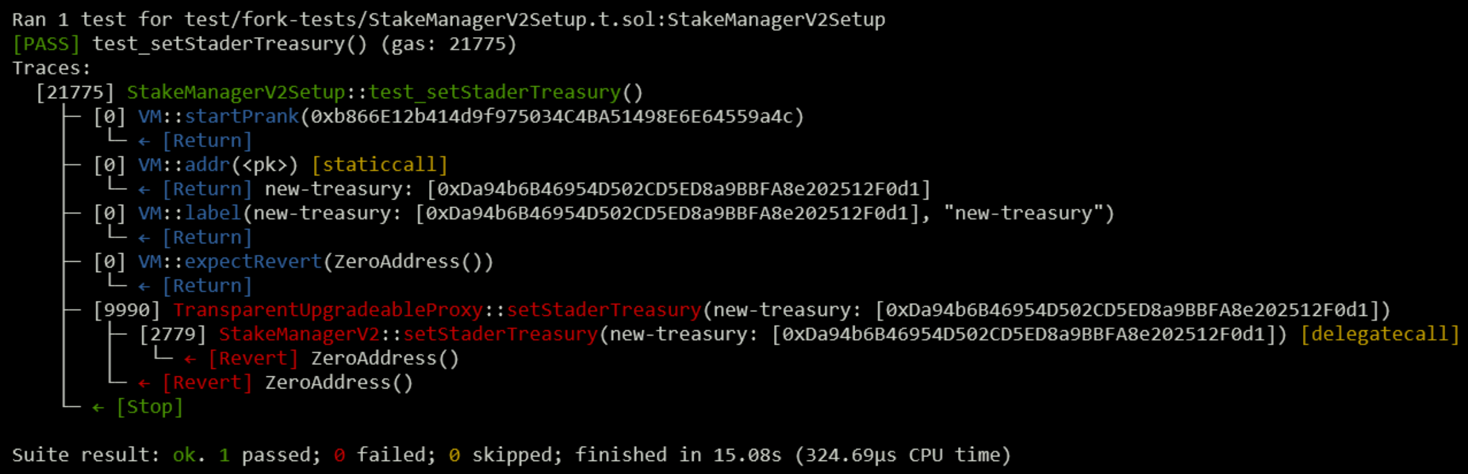
```
if (_staderTreasury != address(0)) revert ZeroAddress();
```

This logic is incorrect, as the function should allow the `_staderTreasury` address to be set to any non-zero address. This bug renders the `setStaderTreasury()` function unusable, potentially disrupting operations that rely on a properly set treasury address.

Proof of Concept

In the following example, the `setStaderTreasury()` function fails on execution:

```
function test_setStaderTreasury() public {
  vm.startPrank(admin);
  address newTreasury = makeAddr("new-treasury");
  vm.expectRevert(ZeroAddress.selector);
  stakeManagerV2.setStaderTreasury(newTreasury);
}
```



```
Ran 1 test for test/fork-tests/StakeManagerV2Setup.t.sol:StakeManagerV2Setup
[PASS] test_setStaderTreasury() (gas: 21775)
Traces:
  [21775] StakeManagerV2Setup::test_setStaderTreasury()
    ├─ [0] VM::startPrank(0xb866E12b414d9f975034C4BA51498E6E64559a4c)
    │   └─ ← [Return]
    ├─ [0] VM::addr(<pk>) [staticcall]
    │   └─ ← [Return] new-treasury: [0xDa94b6B46954D502CD5ED8a9BBFA8e202512F0d1]
    ├─ [0] VM::label(new-treasury: [0xDa94b6B46954D502CD5ED8a9BBFA8e202512F0d1], "new-treasury")
    │   └─ ← [Return]
    ├─ [0] VM::expectRevert(ZeroAddress())
    │   └─ ← [Return]
    ├─ [9990] TransparentUpgradeableProxy::setStaderTreasury(new-treasury: [0xDa94b6B46954D502CD5ED8a9BBFA8e202512F0d1])
    │   ├─ [2779] StakeManagerV2::setStaderTreasury(new-treasury: [0xDa94b6B46954D502CD5ED8a9BBFA8e202512F0d1]) [delegatecall]
    │   │   └─ ← [Revert] ZeroAddress()
    │   └─ ← [Revert] ZeroAddress()
    └─ ← [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 15.08s (324.69µs CPU time)
```

BVSS

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

Recommendation

Refactor the `if` statement to revert only when the new `_staderTreasury` address is the zero address.

```
if (_staderTreasury == address(0)) revert ZeroAddress();
```

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

StakeManagerV2.sol#L277

7.3 INCORRECT LOGIC AFFECTS UNDELEGATION PROCESS

// LOW

Description

The _computeBnbXToBurn function in the StakeManagerV2.sol contract has a complex loop that calculates the amount of BNBX to burn. This function is used in the startBatchUndelegation() function to calculate the amount of BNBX to burn when a user starts batch undelegation. However, the logic in the _computeBnbXToBurn function is incorrect, as it increments the value of the firstUnprocessedUserIndex variable prior to using this value to update the cummulativeBnbXToBurn variable.

```
function _computeBnbXToBurn(
    uint256 _batchSize,
    address _creditContract
)
    internal
    returns (uint256 amountInBnbXToBurn)
{
    uint256 pooledBnb = IStakeCredit(_creditContract).getPooledBNB(address(this));

    uint256 cummulativeBnbXToBurn = amountInBnbXToBurn +
withdrawalRequests[firstUnprocessedUserIndex].amountInBnbX;
    uint256 cummulativeBnbToWithdraw = convertBnbXToBnb(cummulativeBnbXToBurn);
    uint256 processedCount;

    while (
        (processedCount < _batchSize) && (firstUnprocessedUserIndex <
withdrawalRequests.length)
            && (cummulativeBnbToWithdraw <= pooledBnb)
    ) {
        amountInBnbXToBurn = cummulativeBnbXToBurn;
        withdrawalRequests[firstUnprocessedUserIndex].processed = true;
        withdrawalRequests[firstUnprocessedUserIndex].batchId =
batchWithdrawalRequests.length;
        processedCount++;
        firstUnprocessedUserIndex++;
        // below line won't end up in infinite loop, these checks will stop it
        // (processedCount < _batchSize) && (firstUnprocessedUserIndex <
withdrawalRequests.length)
        cummulativeBnbXToBurn +=
withdrawalRequests[firstUnprocessedUserIndex].amountInBnbX;
        cummulativeBnbToWithdraw = convertBnbXToBnb(cummulativeBnbXToBurn);
    }
    if (amountInBnbXToBurn == 0) revert NoWithdrawalRequests();
}
```

This leads to an impossibility of executing the `startBatchUndelegation()` function succesfully.

Proof of Concept

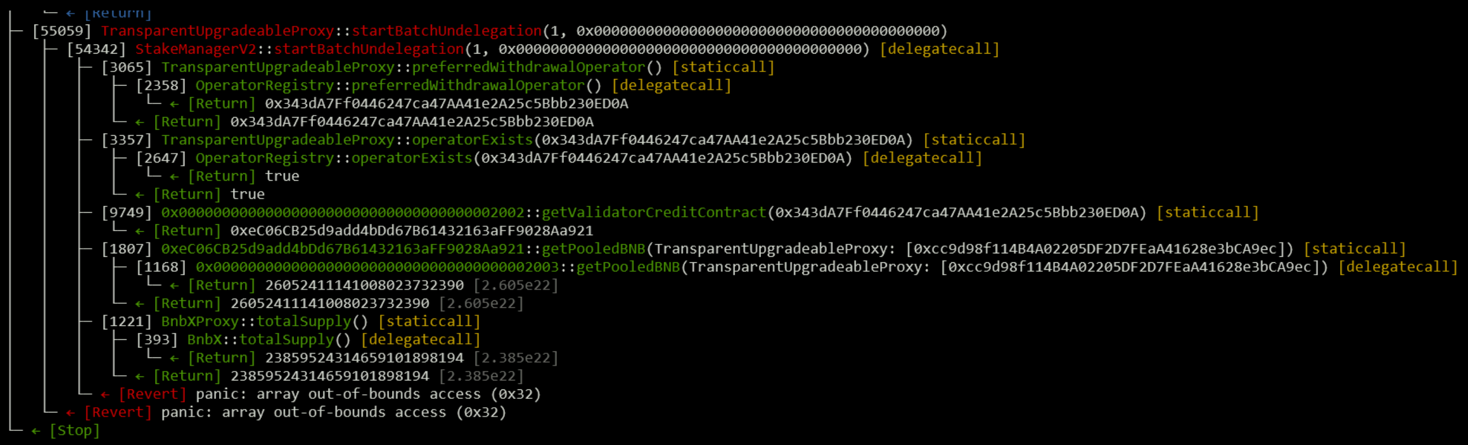
In the following scenario, the `startBatchUndelegation()` function call fails on execution:

```
function test_startBatchUndelegation() public {
  uint256 amount = 1 ether;
  vm.deal(manager, amount * 10);
  deal(address(bnbxAddrProxy), user1, amount * 10);

  // there has to be a withdrawalrequest
  vm.deal(user1, amount);
  vm.startPrank(user1);
  uint256 amountMinted = stakeManagerV2.delegate{value: amount}("referral")
  IERC20(bnbxAddrProxy).approve(address(stakeManagerV2), type(uint256).max)

  stakeManagerV2.requestWithdraw(amountMinted);
  assertEq(stakeManagerV2.getWithdrawalRequestCount(), 1);
  assertEq(IERC20(bnbxAddrProxy).balanceOf(address(stakeManagerV2)), amount
  vm.stopPrank();

  vm.startPrank(staderOperator);
  vm.expectRevert();
  stakeManagerV2.startBatchUndelegation(1, address(0));
}
```



```
│   └─ ← [Return]
├─ [55059] TransparentUpgradeableProxy::startBatchUndelegation(1, 0x0000000000000000000000000000000000000000)
│   ├─ [54342] StakeManagerV2::startBatchUndelegation(1, 0x0000000000000000000000000000000000000000) [delegatecall]
│   │   ├─ [3065] TransparentUpgradeableProxy::preferredWithdrawalOperator() [staticcall]
│   │   │   ├─ [2358] OperatorRegistry::preferredWithdrawalOperator() [delegatecall]
│   │   │   │   └─ ← [Return] 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A
│   │   │   └─ ← [Return] 0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A
│   │   ├─ [3357] TransparentUpgradeableProxy::operatorExists(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [staticcall]
│   │   │   ├─ [2647] OperatorRegistry::operatorExists(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [delegatecall]
│   │   │   │   └─ ← [Return] true
│   │   │   └─ ← [Return] true
│   │   ├─ [9749] 0x0000000000000000000000000000000000002002::getValidatorCreditContract(0x343dA7Ff0446247ca47AA41e2A25c5Bbb230ED0A) [staticcall]
│   │   │   └─ ← [Return] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921
│   │   ├─ [1807] 0xeC06CB25d9add4bDd67B61432163aFF9028Aa921::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [staticcall]
│   │   │   ├─ [1168] 0x0000000000000000000000000000000000002003::getPooledBNB(TransparentUpgradeableProxy: [0xcc9d98f114B4A02205DF2D7FEaA41628e3bCA9ec]) [delegatecall]
│   │   │   │   └─ ← [Return] 26052411141008023732390 [2.605e22]
│   │   │   └─ ← [Return] 26052411141008023732390 [2.605e22]
│   │   ├─ [1221] BnbXProxy::totalSupply() [staticcall]
│   │   │   ├─ [393] BnbX::totalSupply() [delegatecall]
│   │   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   │   └─ ← [Return] 23859524314659101898194 [2.385e22]
│   │   └─ ← [Revert] panic: array out-of-bounds access (0x32)
│   └─ ← [Revert] panic: array out-of-bounds access (0x32)
└─ ← [Stop]
```

BVSS

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

Recommendation

Consider updating the `_computeBnbXToBurn` function to correctly calculate the amount of BNBX to burn by incrementing the `firstUnprocessedUserIndex` variable after updating the `cummulativeBnbXToBurn` variable.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

StakeManagerV2.sol#L376-L379

7.4 INCORRECT NATSPEC

// INFORMATIONAL

Description

The `completeBatchUndelegation()` from the `StakeManagerV2.sol` contract is designed to complete the undelegation process. According to its NATSPEC documentation, the function should only be called by an authorized address with the `OPERATOR_ROLE`:

`This function can only be called by an address with the OPERATOR_ROLE.`.

However, as stated by the **Stader Labs** team, the NATSPEC comments are incorrect and the `completeBatchUndelegation()` is expected to be open. This could lead to confusion and potential misuse of the function.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (1.5)

Recommendation

Update the NATSPEC documentation to accurately reflect the intended functionality of the `completeBatchUndelegation()` function.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation and removing the incorrect NATSPEC documentation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

StakeManagerV2.sol#L192

7.5 POSSIBLE RE-ENTRANCY BEHAVIOR WHEN MIGRATING FUNDS

// INFORMATIONAL

Description

In the `StakeManagerV1` contract, the `migrateFunds()` function is responsible for transferring funds to the `manager` address, using the low-level `call` method.
While this approach is generally safe with an expected trusted `manager` address, it's worth considering potential edge cases. For example, in an unlikely scenario where the `manager` address points to a contract with code that leads to unexpected behavior, the `migrateFunds()` function could be a point of vulnerability to reentrancy to the `StakeManager` contract due to the flexibility of the low-level `call` method.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (1.5)

Recommendation

In case the `manager` address is a contract, ensure that this contract does not contain any code that could potentially call back into the `StakeManagerV1` contract or interact with any other contracts within the system in unexpected ways.
Alternatively, consider implementing a reentrancy protection mechanism to the functions in the `StakeManager` contract, such as OpenZeppelin's `ReentrancyGuard`.

Remediation Plan

ACKNOWLEDGED: The **Stader Labs team** made a business decision to acknowledge this finding and not alter the contracts, stating that:
If manager is comprised, then it would be able to execute many other privileged functions. And we have 2-step process to set a new manager role, so I believe we are good for now.

References

StakeManager.sol#L337

7.6 NON-REENTRANT MODIFIER ORDERING

// INFORMATIONAL

Description

In Solidity, if a function has multiple modifiers, they are executed in the order specified. If checks or logic of modifiers depend on other modifiers, this has to be considered in their ordering.
Several functions of the contracts in scope have multiple modifiers, with one of them being `nonReentrant` which prevents reentrancy behavior on the functions. Ideally, the `nonReentrant` modifier should be the first one to prevent even the execution of other modifiers in case of reentrancy behavior.
While there is currently no obvious vulnerability with `nonReentrant` being the last modifier in the list, placing it first ensures that all other modifiers are executed only if the call is non-reentrant. This is a safer practice and can prevent potential issues in future updates or unforeseen scenarios.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:L/Y:N - (0.5)

Recommendation

Switch modifier order to consistently place the `nonReentrant` modifier as the first one to run so that all other modifiers are executed only if the call is non-reentrant.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

7.7 OPEN TO-DOS

// INFORMATIONAL

Description

Throughout the codebase, there are several open TO-DO comments that have not been addressed and which may indicate that the functionality has not been not completely implemented. This incomplete implementation could lead to unexpected behavior and/or potential vulnerabilities within the contract.

Score

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

Recommendation

Ensure that all TO-DO comments are resolved, with priority in addressing them based on their potential impact on the system's security, stability, and functionality. This may involve completing the functionality as intended, removing the comment if it is no longer relevant, or providing additional context if necessary.

Remediation Plan

ACKNOWLEDGED: The **Stader Labs team** made a business decision to acknowledge this finding and not alter the contracts, stating that:
These TODOs are just steps, assertions to perform manually during deployment so that we do not forget.

References

Migration.s.sol#L42
Migration.s.sol#L76-L81
Migration.s.sol#L88

7.8 FLOATING PRAGMA

// INFORMATIONAL

Description

The `StakeManager`contract currently use floating pragma version `^0.8.0`, which means that the code can be compiled by any compiler version that is greater than or equal to `0.8.0`, and less than `0.9.0`. However, it is recommended that 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.

Score

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

Recommendation

Lock the pragma version to the same version used during development and testing.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

StakeManager.sol#L2

7.9 RISK OF EVM VERSION INCOMPATIBILITY ACROSS CHAINS

// INFORMATIONAL

Description

From Solidity versions `0.8.20` through `0.8.24`, the default target EVM version is set to `Shanghai`, which results in the generation of bytecode that includes `PUSH0` opcodes. Starting with version `0.8.25`, the default EVM version shifts to `Cancun`, introducing new opcodes for transient storage, `TSTORE` and `TLOAD`. In this aspect, it is crucial to select the appropriate EVM version when it's intended to deploy the contracts on networks other than the Ethereum mainnet, which may not support these opcodes. Failure to do so could lead to unsuccessful contract deployments or transaction execution issues.

Score

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

Recommendation

Make sure to specify the target EVM version when using Solidity versions from `0.8.20` and above if deploying to chains that may not support newly introduced opcodes. Additionally, it is crucial to stay informed about the opcode support of different chains to ensure smooth deployment and compatibility.

Remediation Plan

ACKNOWLEDGED: The **Stader Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

StakeManagerV2.sol
OperatorRegistry.sol#L2

7.10 REDUNDANT VERIFICATION

// INFORMATIONAL

Description

In the `removeOperator()`, `setPreferredDepositOperator()`, and `setPreferredWithdrawalOperator()` functions from the `StakeManagerV2` contract, there is a requirement that doesn't allow for the `_operator` address to be the zero address:

```
if (_operator == address(0)) revert ZeroAddress();
```

This may be unnecessary as the currently added operator address cannot be the zero address, since this requirement is checked in the `addOperator()` function, which adds to the fact that the mentioned functions have the `whenOperatorDoesExist(_operator)` modifier.

Score

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

Recommendation

Remove the following line from the aforementioned functions:

```
if (_operator == address(0)) revert ZeroAddress();
```

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

OperatorRegistry.sol#L80
OperatorRegistry.sol#L107
OperatorRegistry.sol#L124

7.11 USE OF MAGIC NUMBERS

// INFORMATIONAL

Description

Throughout the contracts in scope, there are several instances where magic numbers are used. Magic numbers are direct numerical or string values used in the code without any explanation or context. This practice can lead to code maintainability issues, potential errors, and difficulty in understanding the code's logic.

Score

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

Recommendation

To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name. Consider adding an inline comment explaining how the magic numbers are calculated or why they are chosen for complex values.

Remediation Plan

SOLVED: The **Stader Labs team** has addressed the finding in commit `aee951c5477fa8091f25386980fddeac905f9e20` by following the mentioned recommendation.

Remediation Hash

https://github.com/stader-labs/bnbX/commit/aee951c5477fa8091f25386980fddeac905f9e20

References

contracts/StakeManager.sol#L43

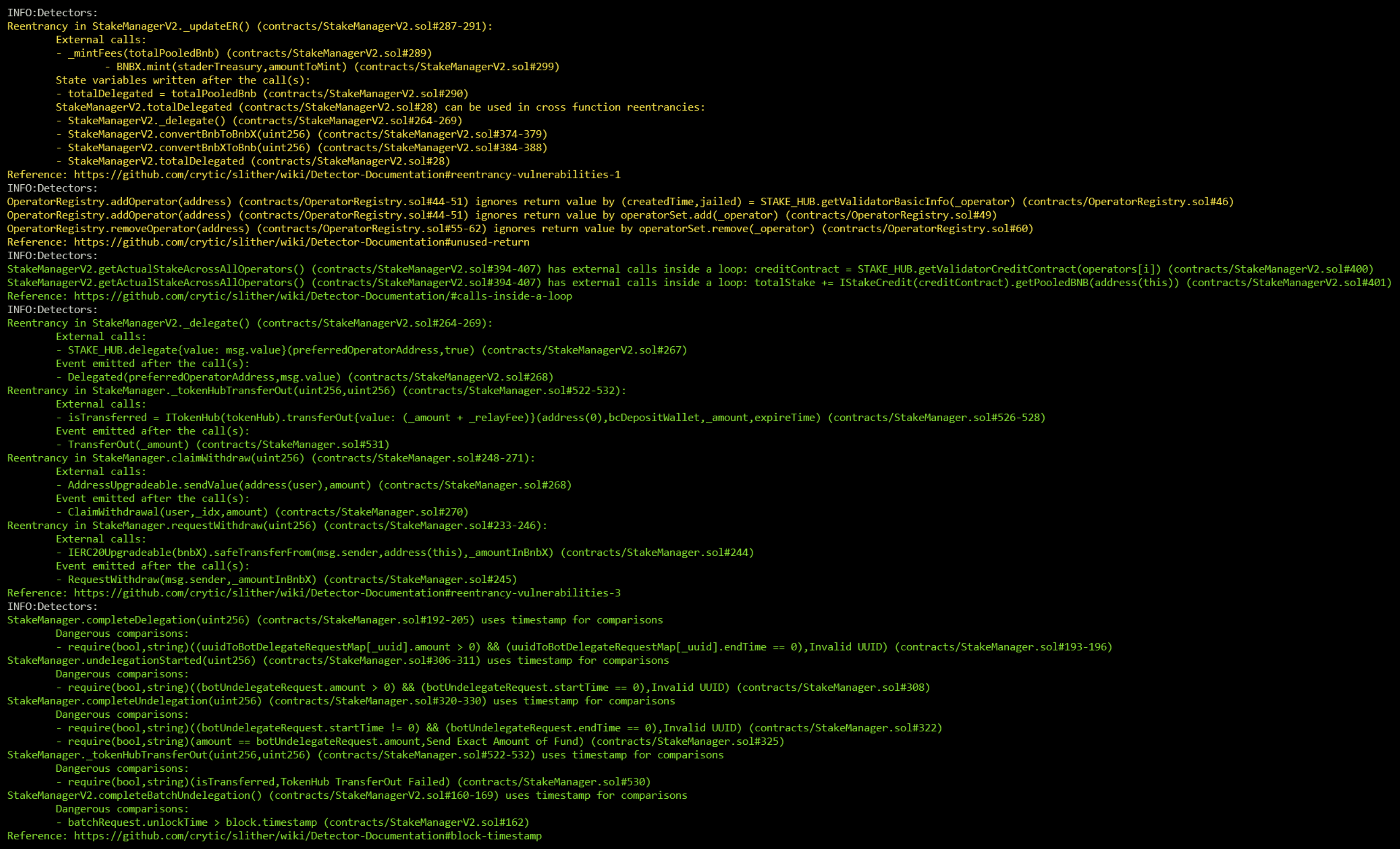
STATIC ANALYSIS REPORT

Description

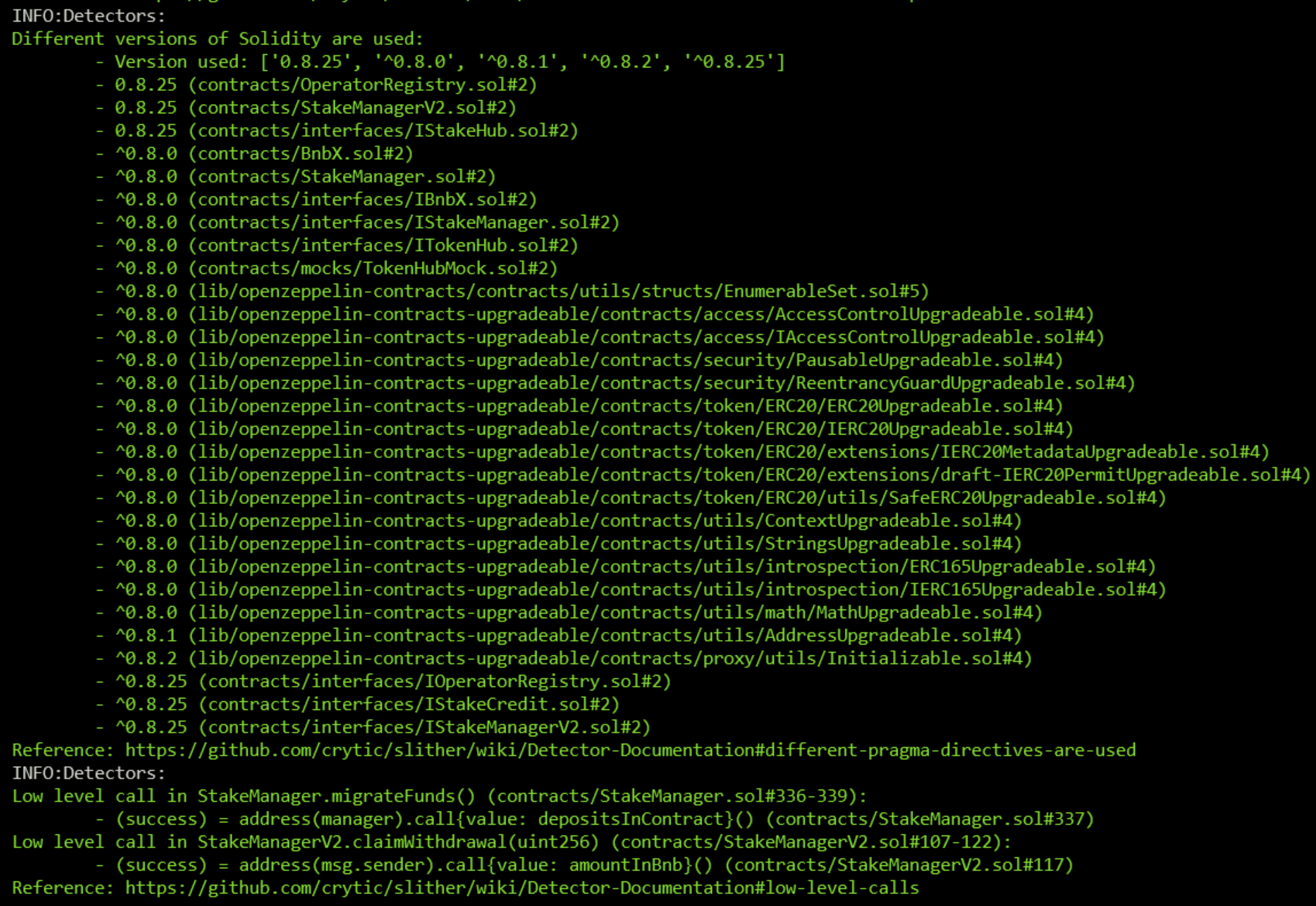
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis 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.
The security team assessed all findings identified by the Slither software, however, findings with related to external dependencies are not included in the below results for the sake of report readability.

Output

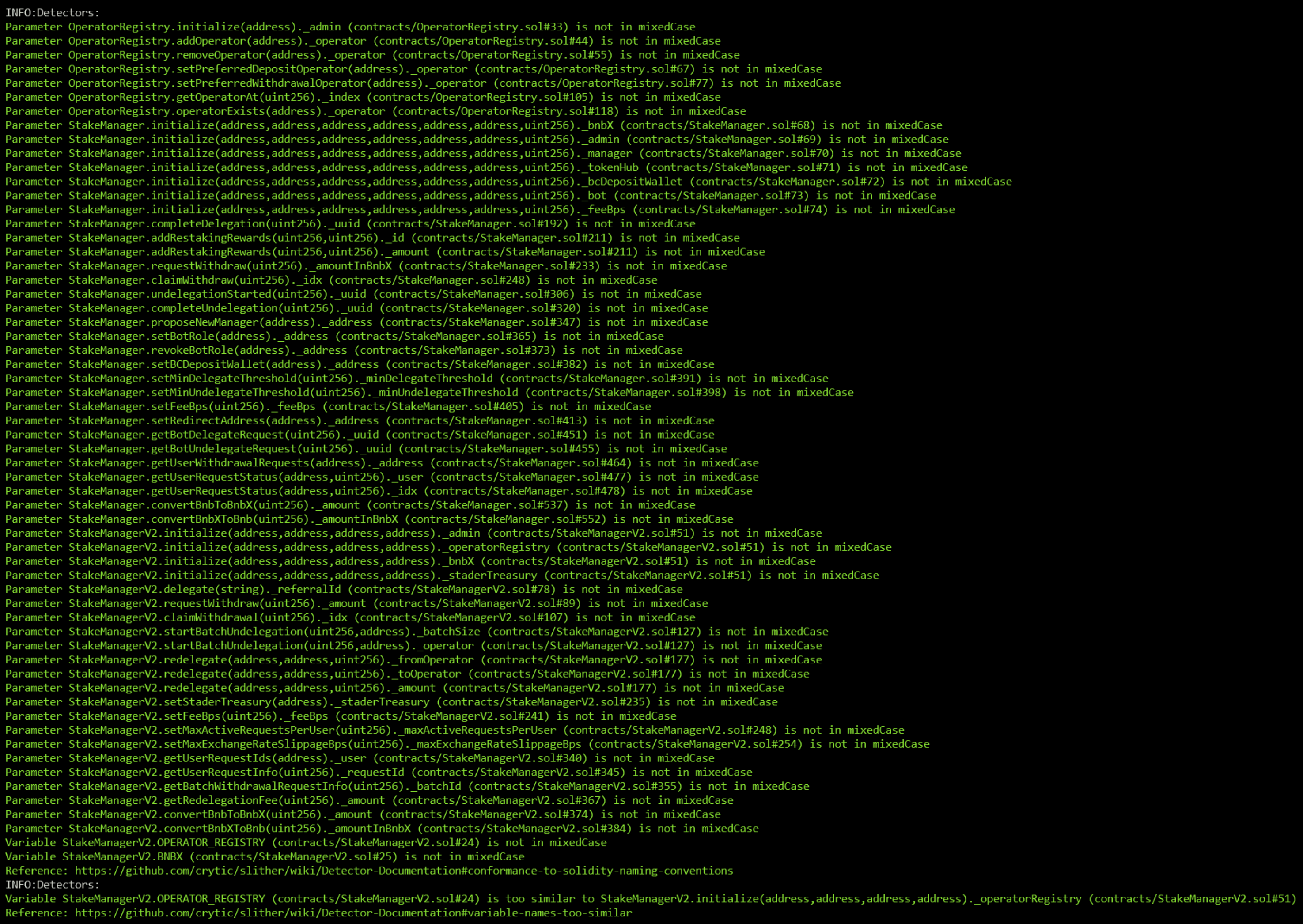
The findings obtained as a result of the Slither scan were reviewed, and most of them were not included in the report because they were determined as false positives.



```
INFO:Detectors:
Reentrancy in StakeManagerV2._updateER() (contracts/StakeManagerV2.sol#287-291):
        External calls:
        - _mintFees(totalPooledBnb) (contracts/StakeManagerV2.sol#289)
                - BNBX.mint(staderTreasury,amountToMint) (contracts/StakeManagerV2.sol#299)
        State variables written after the call(s):
        - totalDelegated = totalPooledBnb (contracts/StakeManagerV2.sol#290)
        StakeManagerV2.totalDelegated (contracts/StakeManagerV2.sol#28) can be used in cross function reentrancies:
        - StakeManagerV2._delegate() (contracts/StakeManagerV2.sol#264-269)
        - StakeManagerV2.convertBnbToBnbX(uint256) (contracts/StakeManagerV2.sol#374-379)
        - StakeManagerV2.convertBnbXToBnb(uint256) (contracts/StakeManagerV2.sol#384-388)
        - StakeManagerV2.totalDelegated (contracts/StakeManagerV2.sol#28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
OperatorRegistry.addOperator(address) (contracts/OperatorRegistry.sol#44-51) ignores return value by (createdTime,jailed) = STAKE_HUB.getValidatorBasicInfo(_operator) (contracts/OperatorRegistry.sol#46)
OperatorRegistry.addOperator(address) (contracts/OperatorRegistry.sol#44-51) ignores return value by operatorSet.add(_operator) (contracts/OperatorRegistry.sol#49)
OperatorRegistry.removeOperator(address) (contracts/OperatorRegistry.sol#55-62) ignores return value by operatorSet.remove(_operator) (contracts/OperatorRegistry.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
StakeManagerV2.getActualStakeAcrossAllOperators() (contracts/StakeManagerV2.sol#394-407) has external calls inside a loop: creditContract = STAKE_HUB.getValidatorCreditContract(operators[i]) (contracts/StakeManagerV2.sol#400)
StakeManagerV2.getActualStakeAcrossAllOperators() (contracts/StakeManagerV2.sol#394-407) has external calls inside a loop: totalStake += IStakeCredit(creditContract).getPooledBNB(address(this)) (contracts/StakeManagerV2.sol#401)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in StakeManagerV2._delegate() (contracts/StakeManagerV2.sol#264-269):
        External calls:
        - STAKE_HUB.delegate{value: msg.value}(preferredOperatorAddress,true) (contracts/StakeManagerV2.sol#267)
        Event emitted after the call(s):
        - Delegated(preferredOperatorAddress,msg.value) (contracts/StakeManagerV2.sol#268)
Reentrancy in StakeManager._tokenHubTransferOut(uint256,uint256) (contracts/StakeManager.sol#522-532):
        External calls:
        - isTransferred = ITokenHub(tokenHub).transferOut{value: (_amount + _relayFee)}(address(0),bcDepositWallet,_amount,expireTime) (contracts/StakeManager.sol#526-528)
        Event emitted after the call(s):
        - TransferOut(_amount) (contracts/StakeManager.sol#531)
Reentrancy in StakeManager.claimWithdraw(uint256) (contracts/StakeManager.sol#248-271):
        External calls:
        - AddressUpgradeable.sendValue(address(user),amount) (contracts/StakeManager.sol#268)
        Event emitted after the call(s):
        - ClaimWithdrawal(user,_idx,amount) (contracts/StakeManager.sol#270)
Reentrancy in StakeManager.requestWithdraw(uint256) (contracts/StakeManager.sol#233-246):
        External calls:
        - IERC20Upgradeable(bnbX).safeTransferFrom(msg.sender,address(this),_amountInBnbX) (contracts/StakeManager.sol#244)
        Event emitted after the call(s):
        - RequestWithdraw(msg.sender,_amountInBnbX) (contracts/StakeManager.sol#245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StakeManager.completeDelegation(uint256) (contracts/StakeManager.sol#192-205) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((uuidToBotDelegateRequestMap[_uuid].amount > 0) && (uuidToBotDelegateRequestMap[_uuid].endTime == 0),Invalid UUID) (contracts/StakeManager.sol#193-196)
StakeManager.undelegationStarted(uint256) (contracts/StakeManager.sol#306-311) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((botUndelegateRequest.amount > 0) && (botUndelegateRequest.startTime == 0),Invalid UUID) (contracts/StakeManager.sol#308)
StakeManager.completeUndelegation(uint256) (contracts/StakeManager.sol#320-330) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((botUndelegateRequest.startTime != 0) && (botUndelegateRequest.endTime == 0),Invalid UUID) (contracts/StakeManager.sol#322)
        - require(bool,string)(amount == botUndelegateRequest.amount,Send Exact Amount of Fund) (contracts/StakeManager.sol#325)
StakeManager._tokenHubTransferOut(uint256,uint256) (contracts/StakeManager.sol#522-532) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(isTransferred,TokenHub TransferOut Failed) (contracts/StakeManager.sol#530)
StakeManagerV2.completeBatchUndelegation() (contracts/StakeManagerV2.sol#160-169) uses timestamp for comparisons
        Dangerous comparisons:
        - batchRequest.unlockTime > block.timestamp (contracts/StakeManagerV2.sol#162)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



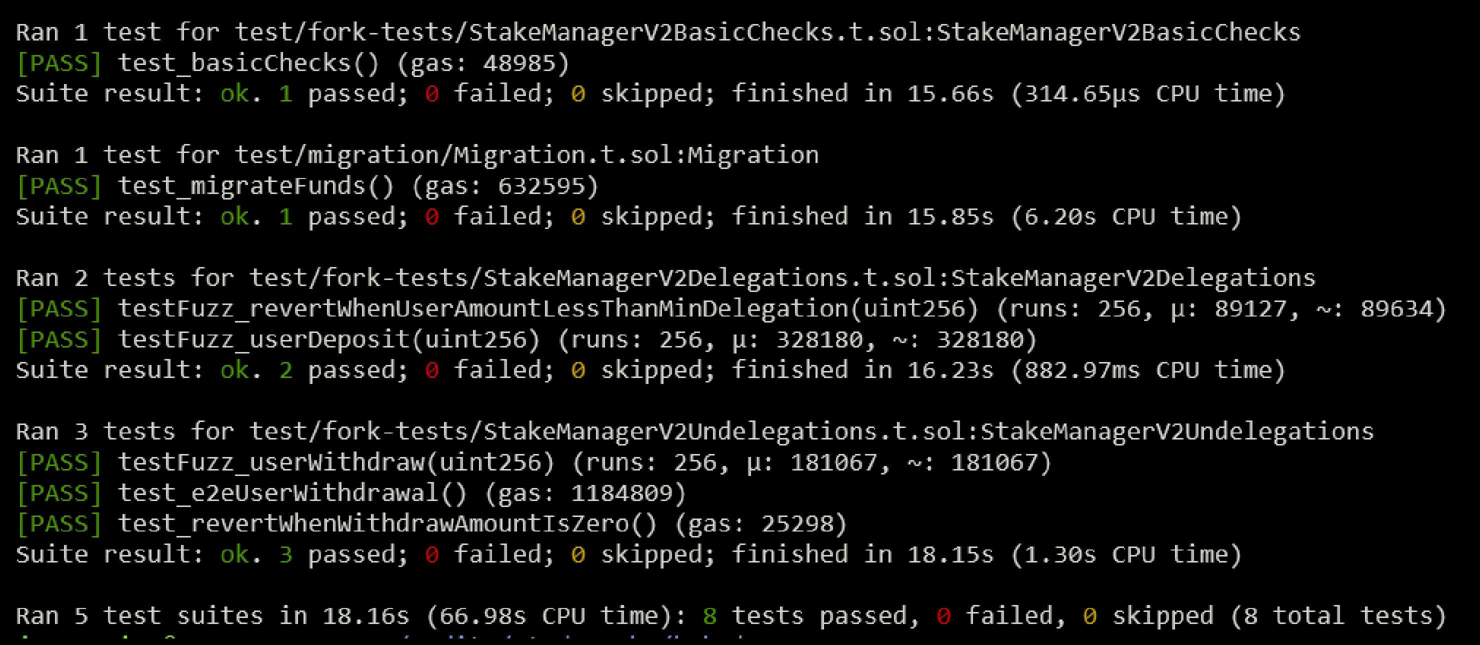
```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.25', '^0.8.0', '^0.8.1', '^0.8.2', '^0.8.25']
        - 0.8.25 (contracts/OperatorRegistry.sol#2)
        - 0.8.25 (contracts/StakeManagerV2.sol#2)
        - 0.8.25 (contracts/interfaces/IStakeHub.sol#2)
        - ^0.8.0 (contracts/BnbX.sol#2)
        - ^0.8.0 (contracts/StakeManager.sol#2)
        - ^0.8.0 (contracts/interfaces/IBnbX.sol#2)
        - ^0.8.0 (contracts/interfaces/IStakeManager.sol#2)
        - ^0.8.0 (contracts/interfaces/ITokenHub.sol#2)
        - ^0.8.0 (contracts/mocks/TokenHubMock.sol#2)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/structs/EnumerableSet.sol#5)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/access/IAccessControlUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/security/PausableUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#4)
        - ^0.8.1 (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#4)
        - ^0.8.2 (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#4)
        - ^0.8.25 (contracts/interfaces/IOperatorRegistry.sol#2)
        - ^0.8.25 (contracts/interfaces/IStakeCredit.sol#2)
        - ^0.8.25 (contracts/interfaces/IStakeManagerV2.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Low level call in StakeManager.migrateFunds() (contracts/StakeManager.sol#336-339):
        - (success) = address(manager).call{value: depositsInContract}() (contracts/StakeManager.sol#337)
Low level call in StakeManagerV2.claimWithdrawal(uint256) (contracts/StakeManagerV2.sol#107-122):
        - (success) = address(msg.sender).call{value: amountInBnb}() (contracts/StakeManagerV2.sol#117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```



```
INFO:Detectors:
Parameter OperatorRegistry.initialize(address)._admin (contracts/OperatorRegistry.sol#33) is not in mixedCase
Parameter OperatorRegistry.addOperator(address)._operator (contracts/OperatorRegistry.sol#44) is not in mixedCase
Parameter OperatorRegistry.removeOperator(address)._operator (contracts/OperatorRegistry.sol#55) is not in mixedCase
Parameter OperatorRegistry.setPreferredDepositOperator(address)._operator (contracts/OperatorRegistry.sol#67) is not in mixedCase
Parameter OperatorRegistry.setPreferredWithdrawalOperator(address)._operator (contracts/OperatorRegistry.sol#77) is not in mixedCase
Parameter OperatorRegistry.getOperatorAt(uint256)._index (contracts/OperatorRegistry.sol#105) is not in mixedCase
Parameter OperatorRegistry.operatorExists(address)._operator (contracts/OperatorRegistry.sol#118) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._bnbX (contracts/StakeManager.sol#68) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._admin (contracts/StakeManager.sol#69) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._manager (contracts/StakeManager.sol#70) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._tokenHub (contracts/StakeManager.sol#71) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._bcDepositWallet (contracts/StakeManager.sol#72) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._bot (contracts/StakeManager.sol#73) is not in mixedCase
Parameter StakeManager.initialize(address,address,address,address,address,address,uint256)._feeBps (contracts/StakeManager.sol#74) is not in mixedCase
Parameter StakeManager.completeDelegation(uint256)._uuid (contracts/StakeManager.sol#192) is not in mixedCase
Parameter StakeManager.addRestakingRewards(uint256,uint256)._id (contracts/StakeManager.sol#211) is not in mixedCase
Parameter StakeManager.addRestakingRewards(uint256,uint256)._amount (contracts/StakeManager.sol#211) is not in mixedCase
Parameter StakeManager.requestWithdraw(uint256)._amountInBnbX (contracts/StakeManager.sol#233) is not in mixedCase
Parameter StakeManager.claimWithdraw(uint256)._idx (contracts/StakeManager.sol#248) is not in mixedCase
Parameter StakeManager.undelegationStarted(uint256)._uuid (contracts/StakeManager.sol#306) is not in mixedCase
Parameter StakeManager.completeUndelegation(uint256)._uuid (contracts/StakeManager.sol#320) is not in mixedCase
Parameter StakeManager.proposeNewManager(address)._address (contracts/StakeManager.sol#347) is not in mixedCase
Parameter StakeManager.setBotRole(address)._address (contracts/StakeManager.sol#365) is not in mixedCase
Parameter StakeManager.revokeBotRole(address)._address (contracts/StakeManager.sol#373) is not in mixedCase
Parameter StakeManager.setBCDepositWallet(address)._address (contracts/StakeManager.sol#382) is not in mixedCase
Parameter StakeManager.setMinDelegateThreshold(uint256)._minDelegateThreshold (contracts/StakeManager.sol#391) is not in mixedCase
Parameter StakeManager.setMinUndelegateThreshold(uint256)._minUndelegateThreshold (contracts/StakeManager.sol#398) is not in mixedCase
Parameter StakeManager.setFeeBps(uint256)._feeBps (contracts/StakeManager.sol#405) is not in mixedCase
Parameter StakeManager.setRedirectAddress(address)._address (contracts/StakeManager.sol#413) is not in mixedCase
Parameter StakeManager.getBotDelegateRequest(uint256)._uuid (contracts/StakeManager.sol#451) is not in mixedCase
Parameter StakeManager.getBotUndelegateRequest(uint256)._uuid (contracts/StakeManager.sol#455) is not in mixedCase
Parameter StakeManager.getUserWithdrawalRequests(address)._address (contracts/StakeManager.sol#464) is not in mixedCase
Parameter StakeManager.getUserRequestStatus(address,uint256)._user (contracts/StakeManager.sol#477) is not in mixedCase
Parameter StakeManager.getUserRequestStatus(address,uint256)._idx (contracts/StakeManager.sol#478) is not in mixedCase
Parameter StakeManager.convertBnbToBnbX(uint256)._amount (contracts/StakeManager.sol#537) is not in mixedCase
Parameter StakeManager.convertBnbXToBnb(uint256)._amountInBnbX (contracts/StakeManager.sol#552) is not in mixedCase
Parameter StakeManagerV2.initialize(address,address,address,address)._admin (contracts/StakeManagerV2.sol#51) is not in mixedCase
Parameter StakeManagerV2.initialize(address,address,address,address)._operatorRegistry (contracts/StakeManagerV2.sol#51) is not in mixedCase
Parameter StakeManagerV2.initialize(address,address,address,address)._bnbX (contracts/StakeManagerV2.sol#51) is not in mixedCase
Parameter StakeManagerV2.initialize(address,address,address,address)._staderTreasury (contracts/StakeManagerV2.sol#51) is not in mixedCase
Parameter StakeManagerV2.delegate(string)._referralId (contracts/StakeManagerV2.sol#78) is not in mixedCase
Parameter StakeManagerV2.requestWithdraw(uint256)._amount (contracts/StakeManagerV2.sol#89) is not in mixedCase
Parameter StakeManagerV2.claimWithdrawal(uint256)._idx (contracts/StakeManagerV2.sol#107) is not in mixedCase
Parameter StakeManagerV2.startBatchUndelegation(uint256,address)._batchSize (contracts/StakeManagerV2.sol#127) is not in mixedCase
Parameter StakeManagerV2.startBatchUndelegation(uint256,address)._operator (contracts/StakeManagerV2.sol#127) is not in mixedCase
Parameter StakeManagerV2.redelegate(address,address,uint256)._fromOperator (contracts/StakeManagerV2.sol#177) is not in mixedCase
Parameter StakeManagerV2.redelegate(address,address,uint256)._toOperator (contracts/StakeManagerV2.sol#177) is not in mixedCase
Parameter StakeManagerV2.redelegate(address,address,uint256)._amount (contracts/StakeManagerV2.sol#177) is not in mixedCase
Parameter StakeManagerV2.setStaderTreasury(address)._staderTreasury (contracts/StakeManagerV2.sol#235) is not in mixedCase
Parameter StakeManagerV2.setFeeBps(uint256)._feeBps (contracts/StakeManagerV2.sol#241) is not in mixedCase
Parameter StakeManagerV2.setMaxActiveRequestsPerUser(uint256)._maxActiveRequestsPerUser (contracts/StakeManagerV2.sol#248) is not in mixedCase
Parameter StakeManagerV2.setMaxExchangeRateSlippageBps(uint256)._maxExchangeRateSlippageBps (contracts/StakeManagerV2.sol#254) is not in mixedCase
Parameter StakeManagerV2.getUserRequestIds(address)._user (contracts/StakeManagerV2.sol#340) is not in mixedCase
Parameter StakeManagerV2.getUserRequestInfo(uint256)._requestId (contracts/StakeManagerV2.sol#345) is not in mixedCase
Parameter StakeManagerV2.getBatchWithdrawalRequestInfo(uint256)._batchId (contracts/StakeManagerV2.sol#355) is not in mixedCase
Parameter StakeManagerV2.getRedelegationFee(uint256)._amount (contracts/StakeManagerV2.sol#367) is not in mixedCase
Parameter StakeManagerV2.convertBnbToBnbX(uint256)._amount (contracts/StakeManagerV2.sol#374) is not in mixedCase
Parameter StakeManagerV2.convertBnbXToBnb(uint256)._amountInBnbX (contracts/StakeManagerV2.sol#384) is not in mixedCase
Variable StakeManagerV2.OPERATOR_REGISTRY (contracts/StakeManagerV2.sol#24) is not in mixedCase
Variable StakeManagerV2.BNBX (contracts/StakeManagerV2.sol#25) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable StakeManagerV2.OPERATOR_REGISTRY (contracts/StakeManagerV2.sol#24) is too similar to StakeManagerV2.initialize(address,address,address,address)._operatorRegistry (contracts/StakeManagerV2.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

UNIT TESTS AND FUZZING

The original repository used the **Foundry** environment to develop and test the smart contracts. All tests were executed successfully. Additionally, the test codebase was amplified to generate additional tests against a local fork that covered. These additional tests were run successfully.



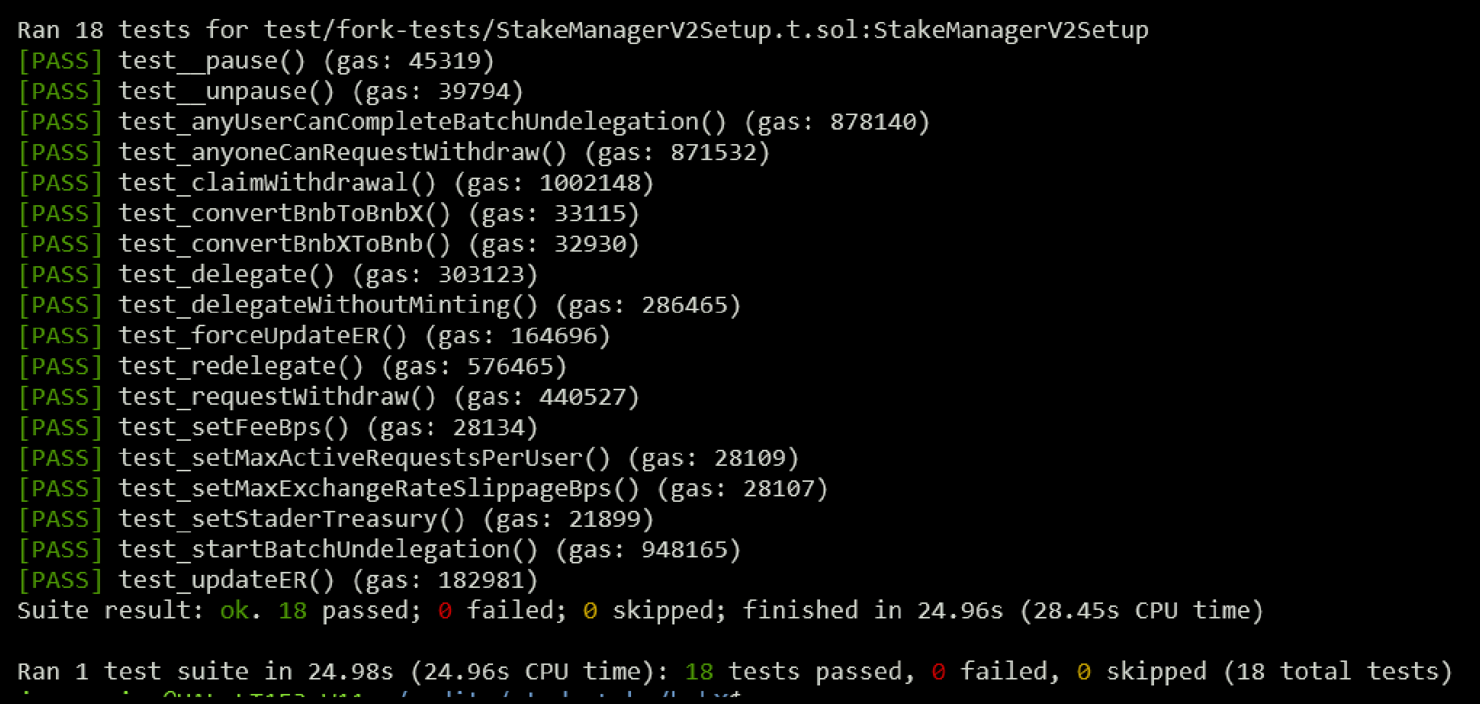
```
Ran 1 test for test/fork-tests/StakeManagerV2BasicChecks.t.sol:StakeManagerV2BasicChecks
[PASS] test_basicChecks() (gas: 48985)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 15.66s (314.65µs CPU time)

Ran 1 test for test/migration/Migration.t.sol:Migration
[PASS] test_migrateFunds() (gas: 632595)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 15.85s (6.20s CPU time)

Ran 2 tests for test/fork-tests/StakeManagerV2Delegations.t.sol:StakeManagerV2Delegations
[PASS] testFuzz_revertWhenUserAmountLessThanMinDelegation(uint256) (runs: 256, µ: 89127, ~: 89634)
[PASS] testFuzz_userDeposit(uint256) (runs: 256, µ: 328180, ~: 328180)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 16.23s (882.97ms CPU time)

Ran 3 tests for test/fork-tests/StakeManagerV2Undelegations.t.sol:StakeManagerV2Undelegations
[PASS] testFuzz_userWithdraw(uint256) (runs: 256, µ: 181067, ~: 181067)
[PASS] test_e2eUserWithdrawal() (gas: 1184809)
[PASS] test_revertWhenWithdrawAmountIsZero() (gas: 25298)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 18.15s (1.30s CPU time)

Ran 5 test suites in 18.16s (66.98s CPU time): 8 tests passed, 0 failed, 0 skipped (8 total tests)
```



```
Ran 18 tests for test/fork-tests/StakeManagerV2Setup.t.sol:StakeManagerV2Setup
[PASS] test__pause() (gas: 45319)
[PASS] test__unpause() (gas: 39794)
[PASS] test_anyUserCanCompleteBatchUndelegation() (gas: 878140)
[PASS] test_anyoneCanRequestWithdraw() (gas: 871532)
[PASS] test_claimWithdrawal() (gas: 1002148)
[PASS] test_convertBnbToBnbX() (gas: 33115)
[PASS] test_convertBnbXToBnb() (gas: 32930)
[PASS] test_delegate() (gas: 303123)
[PASS] test_delegateWithoutMinting() (gas: 286465)
[PASS] test_forceUpdateER() (gas: 164696)
[PASS] test_redelegate() (gas: 576465)
[PASS] test_requestWithdraw() (gas: 440527)
[PASS] test_setFeeBps() (gas: 28134)
[PASS] test_setMaxActiveRequestsPerUser() (gas: 28109)
[PASS] test_setMaxExchangeRateSlippageBps() (gas: 28107)
[PASS] test_setStaderTreasury() (gas: 21899)
[PASS] test_startBatchUndelegation() (gas: 948165)
[PASS] test_updateER() (gas: 182981)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 24.96s (28.45s CPU time)

Ran 1 test suite in 24.98s (24.96s CPU time): 18 tests passed, 0 failed, 0 skipped (18 total tests)
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.