Nexus Network

Smart Contract Security Audit

Prepared by ShellBoxes

Jan 8th, 2024 - Jan 19th, 2024

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Nexus Network
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/Nexus-2023/Nexus-Contracts.git	969e85a8b3d5f5e77ad8453e0a408bfda86bc718
https://github.com/Nexus-2023/off-chain-oracles.git	22b26d3b3da0ff08a7038a0d37d018035359b872

Re-Audit

Repository	Commit Hash
https://github.com/Nexus-2023/Nexus-Contracts	ed6c4d88541b420274028c3c5d1d565d0166cdaf
https://github.com/Nexus-2023/off-chain-oracles	65e5e2237481493be6e14ec0ca383619aecd7f19

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Nexus Network engaged ShellBoxes to conduct a security assessment on the Nexus Network beginning on Jan 8th, 2024 and ending Jan 19th, 2024. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Nexus Network

Nexus Network is building a pluggable solution that enables staking for ETH locked in bridge contracts of rollups.

Issuer	Nexus Network
Website	`https://www.nexusnetwork.co.in`
Type	Solidity Smart Contracts & OffChain Bot Scripts
Documentation	Nexus Network Docs
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Disclaimer

During the audit of the Nexus project, it was noted that the Nexus contracts perform external calls to the SSVNetwork contracts. Additionally, interactions with off-chain components are required, which are explicitly out of scope for this audit and were not reviewed as part of this assessment. Furthermore, it is important to note that the Nexus bridge contracts will be integrated with the rollup contract. Therefore, each rollup project is responsible for ensuring the correct implementation of all features and guaranteeing the security of its project.

On another note, certain features in the off-chain bot scripts were found to be incomplete or not implemented correctly. These issues were identified separately in the audit report.

2.2 Summary

The following is a synopsis of our conclusions from our analysis of the Nexus Network implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.3 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 2 critical-severity, 5 high-severity, 10 medium-severity, 9 low-severity, 1 informational-severity, 1 undetermined-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB-A.1. Public Accessibility of setVariable in Nexus-Library	CRITICAL	Fixed
SHB-B.1. Exposing Private Key Through Command Line Arguments	CRITICAL	Fixed
SHB-A.2. Centralization of Power	HIGH	Acknowledged
SHB-A.3. Potential Desynchronization in rewardsEarned Tracking	HIGH	Acknowledged
SHB-A.4. Admin Rollup Can Manipulate NexusFee	HIGH	Fixed
SHB-B.2. Lack of Deposit Transaction Status Verification	HIGH	Fixed
SHB-B.3. Incomplete Implementation of Functions in Off-chain Code	HIGH	Acknowledged
SHB-A.5. Irreversible Action in addCluster Function Due to Operator ID Error	MEDIUM	Acknowledged
SHB-A.6. Potential Front-Running in setNexusFee Function	MEDIUM	Fixed
SHB-A.7. Potential Precision Loss	MEDIUM	Acknowledged
SHB-A.8. Potential DOS Risk in Initialize Function Due to Uninitialized Addresses	MEDIUM	Mitigated
SHB-A.9. Missing Staking Limit Verification In The changeStakingLimit Function	MEDIUM	Fixed
SHB-A.10. Risk of Incorrect Rollup Validators Count	MEDIUM	Fixed
SHB-A.11. Unverified Validator Exits In The validatorExitBalanceTransferred Function	MEDIUM	Fixed

SHB-A.12. Front-run In The Contract's Initialization	MEDIUM	Mitigated
SHB-B.4. Potential File Content Extraction	MEDIUM	Acknowledged
SHB-B.5. Handling Failed Transactions in Automated Bot Process	MEDIUM	Fixed
SHB-A.13. Unbounded Loops Over Expanding Storage Arrays	LOW	Acknowledged
SHB-A.14. Lack of Validation for SSV Operator Registration	LOW	Acknowledged
SHB-A.15. Reliance on address(this).balance in getRewards	LOW	Acknowledged
SHB-A.16. Potential Reentrancy Attacks in Multiple Functions	LOW	Mitigated
SHB-A.17. Potential Denial of Service (DoS) in redeemRewards	LOW	Acknowledged
SHB-A.18. Missing Address Verification	LOW	Fixed
SHB-A.19. Floating Pragma	LOW	Fixed
SHB-B.6. Inaccurate Local RPC Check in EthNode Class	LOW	Acknowledged
SHB-B.7. Potential Race Condition in get_node_operators Method	LOW	Acknowledged
SHB-A.20. Inconsistent NEXUS_NETWORK Address in Nexus Contracts	INFORMATIONAL	Fixed
SHB-B.8. Inefficiencies and Misnomers in get_latest_nonce Function	UNDETERMINED	Acknowledged

3 Finding Details

A Nexus Contracts

SHB-A.1 Public Accessibility of setVariable in NexusLibrary

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The setVariable function in NexusLibrary is marked as public, allowing any external entity to call it. This function directly modifies contract storage using low-level assembly code with sstore. Given its unrestricted access, malicious actors can manipulate the contract's state, posing a critical security risk.

Files Affected:

SHB-A.1.1: nexusLibrary.sol

```
63      function setVariable(bytes32 _slot, uint256 amount) public {
64          assembly {
65              sstore(_slot, amount)
66          }
67      }
```

Recommendation:

To mitigate this security risk, the accessibility of the setVariable function should be restricted:

- Change the function's visibility from public to internal. This ensures that setVariable can only be called by the NexusLibrary contract or contracts deriving from it, not by external actors.

Updates

The team resolved the issue by modifying the visibility of the setVariable function to internal.

SHB-A.1.2: nexusLibrary.sol

```
69      function setVariable(bytes32 _slot, uint256 amount) internal {
70          assembly {
71              sstore(_slot, amount)
72          }
73      }
```

SHB-A.2 Centralization of Power

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 2
- Impact : 3

Description:

The updateRewardsRollup function and others, which are exclusively callable by rewardBot or OffChainBot, centralizes significant administrative power. This setup poses a risk in terms of security and trust, as the rewardBot and OffChainBot have extensive control over the distribution and updating of rewards, potentially leading to single points of failure or manipulation.

Files Affected:

SHB-A.2.1: ValidatorExecutionRewards.sol

```
53      function updateRewardsRollup(RollupExecutionReward[] calldata
            ↪ rewards) external onlyRewardBot {
54          uint256 total_rewards;
55          for(uint i=0;i<rewards.length;i++){
56              executionRewards[rewards[i].rollupAdmin] += rewards[i].amount
                    ↪ ;
```

```
57              total_rewards+=rewards[i].amount;
58              emit RollupExecutionRewardUpdated(rewards[i].rollupAdmin,
                    ↪ rewards[i].amount);
59          }
60          if (total_rewards>(rewardsEarned-rewardsClaimed)) revert
                ↪ IncorrectRewards();
61      }
```

SHB-A.2.2: Nexus.sol

```
201     function validatorSlashed(address rollupAdmin, uint256 amountSlashed
            ↪ ) external onlyOffChainBot{
202         INexusBridge(rollups[rollupAdmin].bridgeContract).
                ↪ validatorsSlashed(amountSlashed);
203         emit RollupValidatorSlashed(rollupAdmin,amountSlashed);
204     }
```

SHB-A.2.3: Nexus.sol

```
145     function depositValidatorRollup(
146         address _rollupAdmin,
147         Validator[] calldata _validators
148     ) external override onlyOffChainBot {
149         INexusBridge(rollups[_rollupAdmin].bridgeContract)
150             .depositValidatorNexus(
151                 _validators,
152                 uint256(rollups[_rollupAdmin].stakingLimit)
153                 );
154         for (uint i = 0; i < _validators.length; i++) {
155             depositingPubkeys.addElement(_validators[i].pubKey);
156             emit ValidatorSubmitted(_validators[i].pubKey, _rollupAdmin);
157         }
158     }
```

SHB-A.2.4: Nexus.sol

```
160     function depositValidatorShares(
161         address _rollupAdmin,
162         ValidatorShares calldata _validatorShare
163     ) external override onlyOffChainBot {
164         (bool key_present, uint256 index) = depositingPubkeys.findElement
                ↪ (
165             _validatorShare.pubKey
166         );
167         if (!key_present) revert KeyNotDeposited();
168         IERC20(SSV_TOKEN).approve(SSV_NETWORK, _validatorShare.amount);
169         ISSVNetworkCore(SSV_NETWORK).registerValidator(
170             _validatorShare.pubKey,
171             _validatorShare.operatorIds,
172             _validatorShare.sharesEncrypted,
173             _validatorShare.amount,
174             _validatorShare.cluster
175         );
176         depositingPubkeys.removeElement(_validatorShare.pubKey);
177         activePubkeys.addElement(_validatorShare.pubKey);
178         emit ValidatorShareSubmitted(_validatorShare.pubKey, _rollupAdmin
                ↪ ,_validatorShare.amount);
179     }
```

SHB-A.2.5: Nexus.sol

```
181     function validatorExit(address rollupAdmin,bytes[] calldata pubkeys)
            ↪  external onlyOffChainBot{
182         for(uint i=0;i<pubkeys.length;i++){
183             (bool key_present, uint256 index) = activePubkeys.findElement
                    ↪ (pubkeys[i]);
184             if (key_present){
185                 activePubkeys.removeElement(pubkeys[i]);
186                 exitingKeys.addElement(pubkeys[i]);
187                 emit ValidatorExitSubmitted(rollupAdmin,pubkeys[i]);
```

```
188             }else{
189                 revert InvalidKeySupplied();
190             }
191         }
192     }
```

SHB-A.2.6: Nexus.sol

```
194     function validatorExitBalanceTransferred(address rollupAdmin,bytes
            ↪ calldata pubkey, uint64[] memory operatorIds, ISSVNetworkCore.
            ↪ Cluster memory cluster) external onlyOffChainBot{
195         ISSVNetworkCore(SSV_NETWORK).removeValidator(pubkey, operatorIds,
                ↪  cluster);
196         exitingKeys.removeElement(pubkey);
197         emit ValidatorExited(rollupAdmin,pubkey);
198         INexusBridge(rollups[rollupAdmin].bridgeContract).
                ↪ updateExitedValidators();
199     }
```

Recommendation:

- Explore options such as multi-signature requirements for critical actions or decentralized governance models where multiple stakeholders have a say in key decisions.
- Consider making the rewardBot and OffChainBot decentralized and have a voting mechanism for reaching a consensus.

Updates

The Nexus team has acknowledged this issue and has confirmed plans to address it by decentralizing the rewardBot and OffChainBot. The implementation is scheduled for the upcoming version.

SHB-A.3 Potential Desynchronization in rewardsEarned Tracking

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 2
- Impact : 3

Description:

The current implementation uses the rewardsEarned variable to track rewards, incrementing it within a receive() function when Ether is sent to the contract. This approach poses a risk of desynchronization, as rewardsEarned could be artificially inflated through either malicious or unintentional Ether transfers. External parties could manipulate the contract's state by sending Ether, leading to discrepancies in reward tracking and potential vulnerabilities in reward distribution logic.

Files Affected:

SHB-A.3.1: ValidatorExecutionRewards.sol

```
44      receive() external payable {
45          rewardsEarned+=msg.value;
46          emit ExecutionRewardsReceived(msg.value);
47      }
```

Recommendation:

To mitigate this risk, it's recommended to replace the receive() function with a controlled, payable function that includes appropriate access control mechanisms. This function should be the only way to increment rewardsEarned, ensuring that all Ether transfers contributing to the rewards are intentional and authorized. By doing so, the contract will not increment rewardsEarned if Ether is forcefully sent to it, preventing potential desynchronization. Removing or limiting the receive() function's ability to impact critical contract variables is crucial for maintaining the integrity of the rewards tracking system.

Additionally, implementing checks or constraints within the new payable function to validate the transactions can further enhance security and control.

Updates

The Nexus team has acknowledged the issue and opted to retain the receive() function to facilitate the reception of proposer rewards. Furthermore, they have stated that any additional assets sent by a third party will still remain unaccounted for and will not affect the contract's operation.

SHB-A.4 Admin Rollup Can Manipulate NexusFee

- Severity : HIGH
- Status : Fixed
- Likelihood : 2
- Impact : 3

Description:

The Nexus contract currently permits rollup admins to freely change the NexusFee, potentially manipulating the NexusFeePercentage by decreasing its value at any time. This manipulation affects redeemRewards fee calculations, possibly leaving the Nexus contract without sufficient rewards to cover bot fees. This situation creates a risk where the rollup project could receive most, if not all, of the fees.

Files Affected:

SHB-A.4.1: Nexus.sol

```
131     function changeNexusFee(uint256 _new_fee) external
            ↪ onlyWhitelistedRollup{
132         INexusBridge(rollups[msg.sender].bridgeContract).setNexusFee(
                ↪ _new_fee);
133         emit NexusFeeChanged(msg.sender,_new_fee);
134     }
```

SHB-A.4.2: NexusBridgeDAO.sol

```
28      function redeemRewards(address reward_account) external onlyDAO {
29          uint256 total_rewards = getRewards();
30          if(total_rewards > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
31          uint256 _nexus_rewards = (NexusFeePercentage*total_rewards)/
                ↪ BASIS_POINT;
32          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
33              value: _nexus_rewards,
34              gas: 5000
35          }("");
36          if (nexus_success) {
37              emit NexusRewardsRedeemed(_nexus_rewards);
38          }
39          (bool dao_success, bytes memory dao_data) = reward_account.call{
40              value: (total_rewards - _nexus_rewards),
41              gas: 5000
42          }("");
43          rewardsClaimed += total_rewards;
```

Recommendation:

To address this issue, consider implementing a lower limit fee check for the new NexusFee.

Alternatively, you can design an architecture where rollup admins submit a request to modify the Nexus fee, and only the Nexus owner can approve and accept the request. This adjustment enhances the security of altering the NexusFee.

Updates

The team has addressed the issue by implementing a lower limit fee check for the new NexusFee.

SHB-A.4.3: nexusLibrary.sol

```
63      modifier validNexusFee(uint256 _nexus_fee) {
```

```
64          if (_nexus_fee > (BASIS_POINT) / 10  _nexus_fee <= (BASIS_POINT)
                ↪ / 20)
65              revert IncorrectNexusFee();
66          _;
67      }
```

SHB-A.5 Irreversible Action in addCluster Function Due to Operator ID Error

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The addCluster function risks irreversible actions if the owner mistakenly provides incorrect operator IDs. This function adds an array of operator IDs to a cluster but lacks a mechanism to rectify errors post-addition, potentially leading to operational difficulties.

Files Affected:

SHB-A.5.1: NodeOperator.sol

```
64      function addCluster(
65          uint64[] calldata operatorIds,
66          uint64 clusterId
67      ) external onlyOwner {
68          if (ssvClusters[clusterId].length != 0) revert
                ↪ ClusterAlreadyExited();
69          for (uint256 i=0;i<operatorIds.length;i++){
70              bytes memory ip = bytes(ssvDKGIP[operatorIds[i]]);
71              if (ip.length == 0) revert OperatorNotRegistered();
72          }
```

```
73          ssvClusters[clusterId] = operatorIds;
74          emit ClusterAdded(clusterId, operatorIds);
75      }
```

Recommendation:

Introduce a function to modify or remove operators from a cluster, allowing the owner to correct errors. The steps include:

- Modification Function: Implement a updateClusterOperators function for the owner to change operator IDs in an existing cluster. Ensure this function validates the new operator IDs.
- Enhanced Validation in addCluster: Add more comprehensive validation in the addCluster function to reduce the risk of inputting incorrect operator IDs.

Updates

The Nexus team has acknowledged this issue, stating that updating clusters will make the code complex. Recognizing this complexity, they propose an alternative solution. Instead of modifying clusters, Nexus Network will actively monitor node operator performance and enforce the creation of new clusters in case of malfunctions.

SHB-A.6 Potential Front-Running in setNexusFee Function

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

In the current implementation, the setNexusFee function allows the BridgeDAO to modify the NexusFee. However, if the redeemRewards function in NexusBridgeDAO is called before setNexusFee executes, it can lead to front-running issues.

This scenario may result in redeemRewards processing with an unexpected NexusFee value, causing discrepancies in calculations or distributions.

Files Affected:

SHB-A.6.1: NexusBridgeDAO.sol

```
28      function redeemRewards(address reward_account) external onlyDAO {
29          uint256 total_rewards = getRewards();
30          if(total_rewards > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
31          uint256 _nexus_rewards = (NexusFeePercentage*total_rewards)/
                ↪ BASIS_POINT;
32          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
33              value: _nexus_rewards,
34              gas: 5000
35          }("");
```

Recommendation:

To mitigate front-running risks, modify the redeemRewards function to include an additional parameter, expectedNexusFee. Implement a check at the beginning of redeemRewards to compare this parameter with the current NexusFee. If they don't match, the function should revert. This approach ensures that redeemRewards only processes transactions with the anticipated NexusFee, preventing unexpected outcomes due to changes made by setNexusFee.

- Update redeemRewards Signature: Add uint256 expectedNexusFee as a parameter to redeemRewards.
- Implement Check: At the start of redeemRewards, add:require(expectedNexusFee == NexusFeePercentage, "Unexpected NexusFee");

Updates

The Nexus team has fixed this issue by implementing our recommendation. They have added an expectedFee parameter to the redeemRewards function and ensured that this fee should be equal to the NexusFeePercentage variable.

SHB-A.7 Potential Precision Loss

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 2
- Impact : 2

Description:

The redeemRewards function calculates _nexus_rewards using integer division, which may result in precision loss due to the division by BASIS_POINT (10000). This can lead to inaccuracies in the distribution of rewards, especially when dealing with large numbers or small reward fractions. The same issue also occurs in the updateCValue function in the NexusBridgeUserCValue contract and rebase function located in NexusBridgeUserRebase contract..

Files Affected:

SHB-A.7.1: NexusBridgeDAO.sol

```
28    function redeemRewards(address reward_account) external onlyDAO {
29        uint256 total_rewards = getRewards();
30        if(total_rewards > VALIDATOR_DEPOSIT) revert
              ↪ WaitingForValidatorExits();
31        uint256 _nexus_rewards = (NexusFeePercentage*total_rewards)/
              ↪ BASIS_POINT;
```

SHB-A.7.2: NexusBridgeUserCValue.sol

```
23    function updateCValue() external {
```

```
24          uint256 rewards_to_claim = getRewards() - amountDistributed;
25          if(rewards_to_claim > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
26          uint256 _nexus_rewards = (NexusFeePercentage*rewards_to_claim)/
                ↪ BASIS_POINT;
27          amountDistributed+=rewards_to_claim-_nexus_rewards;
28          cValue = ((amountDeposited - amountWithdrawn)*CValueBasisPoint)
                ↪ /((address(this).balance-_nexus_rewards)+(validatorCount*
                ↪ VALIDATOR_DEPOSIT));
29          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
30              value: _nexus_rewards,
31              gas: 5000
32          }("");
33          if (nexus_success) {
34              emit NexusRewardsRedeemed(_nexus_rewards);
35          }
36          emit CValueUpdated(cValue);
37      }
```

SHB-A.7.3: NexusBridgeUserRebase.sol

```
25      function rebase() external {
26          uint256 rewards_to_claim = getRewards() - amountDistributed;
27          if(rewards_to_claim > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
28          uint256 _nexus_rewards = (NexusFeePercentage*rewards_to_claim)/
                ↪ BASIS_POINT;
```

Recommendation:

To mitigate potential precision loss, consider altering the calculation method to minimize rounding errors:

- Use a higher precision for intermediate calculations before rounding down to the nearest integer. This can be achieved by utilizing fixed-point arithmetic libraries.

Updates

The Nexus team has acknowledged the issue, stating that the observed precision loss is minimal and does not have a significant impact on calculations.

SHB-A.8 Potential DOS Risk in Initialize Function Due to Uninitialized Addresses

- Severity : MEDIUM
- Likelihood : 1
- Status : Mitigated
- Impact : 3

Description:

The initialize function within the Nexus contract, crucial for upgradable contract initialization, introduces a substantial potential Denial-of-Service (DOS) risk. The critical addresses, specifically the NodeOperatorContract and offChainBot addresses, remain uninitialized. Due to the contract's upgradable nature, initializing the offChainBot address in the contract declaration is not valid, and it persists as address(0). This creates a serious threat as it blocks all offchain bot actions. Additionally, rollup admins are unable to register a rollup or change the cluster until the owner sets these addresses using the setNodeOperatorContract and setOffChainBot functions.

Files Affected:

SHB-A.8.1: Nexus.sol

```
32      address public offChainBot = 0
            ↪ x45a3f77543167c8D0965194879c4e0B0dbB581d0;
```

SHB-A.8.2: Nexus.sol

```
38      address public NodeOperatorContract;
```

SHB-A.8.3: Nexus.sol

```
58      function initialize() public initilizeOnce {
59          _ownableInit(msg.sender);

61      }
```

Recommendation:

To address this issue, consider initializing the NodeOperatorContract and offChainBot addresses in the initialize function of the Nexus contract.

Updates

The Nexus team has mitigated this risk, and they have confirmed that the required addresses, including the NodeOperatorContract and offChainBot, will be initialized directly through the deployment scripts.

SHB-A.9 Missing Staking Limit Verification In The changeStakingLimit Function

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The changeStakingLimit function in the Nexus contract lacks proper verification for the newStakingLimit variable to ensure it is less than or equal to BASIS_POINT. This absence of verification exposes a potential security vulnerability, granting rollup admins the ability to set a rollup staking limit beyond the defined threshold.

Files Affected:

SHB-A.9.1: Nexus.sol

```
121     function changeStakingLimit(
122         uint16 newStakingLimit
123     ) external onlyWhitelistedRollup {
124         rollups[msg.sender].stakingLimit = newStakingLimit;
125         emit StakingLimitChanged(
126             msg.sender,
127             newStakingLimit
128         );
129     }
```

Recommendation:

To address this issue, consider implementing the necessary verification within the function, such as: `if (stakingLimit>BASIS_POINT) revert IncorrectStakingLimit();`. This addition ensures that the new staking limit is within the specified bounds and preventing potential misuse.

Updates

The Nexus team has resolved this issue by adding a verification check in the `changeStakingLimit` function to prevent the new staking limit from exceeding the `BASIS_POINT` threshold.

SHB-A.9.2: Nexus.sol

```
122     function changeStakingLimit(
123         uint16 newStakingLimit
124     ) external onlyWhitelistedRollup {
125         if (newStakingLimit>BASIS_POINT) revert IncorrectStakingLimit();
126         rollups[msg.sender].stakingLimit = newStakingLimit;
```

SHB-A.10 Risk of Incorrect Rollup Validators Count

- Severity : MEDIUM
- Likelihood : 1
- Status : Fixed
- Impact : 3

Description:

The depositValidatorRollup function in the Nexus contract manages the logic for depositing validators. However, it lacks proper checks for duplicate validators. The function adds all keys to the depositingPubkeys array without verifying duplicates, and calls the depositValidatorNexus from the NexusLibrary contract. This flaw increments the number of validators, impacting the validator count and leading to incorrect rewards calculations in the getRewards function.

Files Affected:

SHB-A.10.1: Nexus.sol

```
145     function depositValidatorRollup(
146         address _rollupAdmin,
147         Validator[] calldata _validators
148     ) external override onlyOffChainBot {
149         INexusBridge(rollups[_rollupAdmin].bridgeContract)
150             .depositValidatorNexus(
151                 _validators,
152                 uint256(rollups[_rollupAdmin].stakingLimit)
153                 );
154         for (uint i = 0; i < _validators.length; i++) {
155             depositingPubkeys.addElement(_validators[i].pubKey);
156             emit ValidatorSubmitted(_validators[i].pubKey, _rollupAdmin);
157         }
```

SHB-A.10.2: nexusLibrary.sol

```
90      function depositValidatorNexus(
91          INexusInterface.Validator[] calldata _validators,
92          uint256 stakingLimit
93      ) external onlyNexus {
94          for (uint i = 0; i < _validators.length; i++) {
95              bytes memory withdrawalFromCred = _validators[i]
96                  .withdrawalAddress[12:];
97              if (
98                  keccak256(withdrawalFromCred) !=
99                  keccak256(abi.encodePacked(address(this)))
100             ) revert IncorrectWithdrawalCredentials();
101         }
102         uint256 validatorCount = getVariable(VALIDATOR_COUNT_SLOT);
103         if (
104             (((validatorCount + _validators.length) *
105                 (VALIDATOR_DEPOSIT) *
106                 BASIS_POINT) /
107                 (address(this).balance +
108                     (validatorCount + _validators.length) *
109                     (VALIDATOR_DEPOSIT))) > stakingLimit
110         ) revert StakingLimitExceeding();

112         for (uint i = 0; i < _validators.length; i++) {
113             IDepositContract(DEPOSIT_CONTRACT).deposit{
114                 value: VALIDATOR_DEPOSIT
115             }(
116                 _validators[i].pubKey,
117                 _validators[i].withdrawalAddress,
118                 _validators[i].signature,
119                 _validators[i].depositRoot
120             );
121         }
122         validatorCount += _validators.length;
```

SHB-A.10.3: nexusLibrary.sol

```
130     function getRewards() public view returns (uint256) {
131         uint256 validatorCount = getVariable(VALIDATOR_COUNT_SLOT);
132         uint256 amountDeposited = getVariable(AMOUNT_DEPOSITED_SLOT);
133         uint256 amountWithdrawn = getVariable(AMOUNT_WITHDRAWN_SLOT);
134         uint256 slashedAmount = getVariable(AMOUNT_SLASHED_SLOT);
135         return
136             (address(this).balance + (validatorCount * VALIDATOR_DEPOSIT)
                ↪ ) -
137             (amountDeposited - amountWithdrawn) -
138             slashedAmount;
139     }
```

SHB-A.10.4: NexusBaseBridge.sol

```
53     function depositValidatorNexus(
54         INexusInterface.Validator[] calldata _validators,
55         uint256 stakingLimit
56     ) external override onlyNexus {
57         for (uint i = 0; i < _validators.length; i++) {
58             bytes memory withdrawalFromCred = _validators[i]
59                 .withdrawalAddress[12:];
60             if (
61                 keccak256(withdrawalFromCred) !=
62                 keccak256(abi.encodePacked(address(this)))
63             ) revert IncorrectWithdrawalCredentials();
64         }
65         if (
66             (((validatorCount + _validators.length) *
67                 (VALIDATOR_DEPOSIT) *
68                 BASIS_POINT) /
69                 (address(this).balance +
70                     (validatorCount + _validators.length) *
71                     (VALIDATOR_DEPOSIT))) > stakingLimit
72         ) revert StakingLimitExceeding();
```

```
74          for (uint i = 0; i < _validators.length; i++) {
75              IDepositContract(DEPOSIT_CONTRACT).deposit{
76                  value: VALIDATOR_DEPOSIT
77              }(
78                  _validators[i].pubKey,
79                  _validators[i].withdrawalAddress,
80                  _validators[i].signature,
81                  _validators[i].depositRoot
82              );
83          }
84          validatorCount+=_validators.length;
```

Recommendation:

To address this issue, implement a verification step within the depositValidatorRollup function to ensure that duplicate validators are not added to the depositing keys array.

Updates

The Nexus team has addressed this issue by changing the validators from an array to a mapping structure, associating each pubKey with a ValidatorStatus. The depositValidatorRollup function now checks the status of each key and reverts if a duplicate key exists.

SHB-A.10.5: Nexus.sol

```
32      mapping(bytes=>ValidatorStatus) public validators;
```

SHB-A.10.6: Nexus.sol

```
147     function depositValidatorRollup(
148         address _rollupAdmin,
149         Validator[] calldata _validators
150     ) external override onlyOffChainBot {
151         for (uint i = 0; i < _validators.length; i++) {
152             if(validators[_validators[i].pubKey]!=ValidatorStatus.
                    ↪ INACTIVE) revert IncorrectValidatorStatus();
```

```
153             validators[_validators[i].pubKey] = ValidatorStatus.DEPOSITED
                    ↪ ;
154             emit ValidatorSubmitted(_validators[i].pubKey, _rollupAdmin);
155         }
```

SHB-A.11 Unverified Validator Exits In The validatorExitBalanceTransferred Function

- Severity : MEDIUM
- Likelihood : 1
- Status : Fixed
- Impact : 3

Description:

The validatorExitBalanceTransferred function in the Nexus contract poses a risk by allowing the removal of validators without verifying whether the validatorExit function has been called. Currently, the implementation lacks a crucial check to confirm the existence of the provided pubkey in the exitingKeys array, potentially leading to unintended validator removals.

Files Affected:

SHB-A.11.1: Nexus.sol

```
194     function validatorExitBalanceTransferred(address rollupAdmin,bytes
            ↪ calldata pubkey, uint64[] memory operatorIds, ISSVNetworkCore.
            ↪ Cluster memory cluster) external onlyOffChainBot{
195         ISSVNetworkCore(SSV_NETWORK).removeValidator(pubkey, operatorIds,
                ↪  cluster);
196         exitingKeys.removeElement(pubkey);
197         emit ValidatorExited(rollupAdmin,pubkey);
198         INexusBridge(rollups[rollupAdmin].bridgeContract).
                ↪ updateExitedValidators();
```

```
199     }
```

Recommendation:

Consider implementing a verification step before calling removeValidator to confirm that the pubkey exists in the exitingKeys array.

Updates

The Nexus team has resolved the issue by adding a check for validator exit in the validatorExitBalanceTransferred function. The status of the validator must now be set to VALIDATOR_EXIT_SUBMITTED before any updates are made, aligning with the recommended verification step.

SHB-A.11.2: Nexus.sol

```
187     function validatorExitBalanceTransferred(address rollupAdmin,bytes
            ↪ calldata pubkey, uint64[] memory operatorIds, ISSVNetworkCore.
            ↪ Cluster memory cluster) external onlyOffChainBot{
188         if(validators[pubkey]!=ValidatorStatus.VALIDATOR_EXIT_SUBMITTED)
                ↪ revert IncorrectValidatorStatus();
189         ISSVNetworkCore(SSV_NETWORK).removeValidator(pubkey, operatorIds,
                ↪  cluster);
190         INexusBridge(rollups[rollupAdmin].bridgeContract).
                ↪ updateExitedValidators();
```

SHB-A.12 Front-run In The Contract's Initialization

- Severity : MEDIUM
- Likelihood : 1
- Status : Mitigated
- Impact : 3

Description:

The initialize function in both the NodeOperator and Nexus contracts poses a potential front-run attack risk, potentially allowing an attacker to take ownership of the contracts. As the function is designed to be called once during initialization, the lack of additional security measures creates a window of vulnerability where an attacker can front-run the initialization transaction and set themselves as the owner.

Exploit Scenario:

The owner deploys the contract and performs the initialize function, then the attacker front-runs the transaction by submitting a transaction with a higher gas price, enabling them to take ownership of the contract

Files Affected:

SHB-A.12.1: Nexus.sol

```
26 contract Nexus is INexusInterface, Ownable, Proxiable {
```

SHB-A.12.2: NodeOperator.sol

```
17 contract NodeOperator is Ownable, Proxiable, INodeOperator{
```

Recommendation:

Consider deploying the contract and initializing it in the same transaction. Alternatively, implement the logic of the upgradeToAndCall function to initialize the implementation contract within the same transaction. This approach aligns with best practices and can be seamlessly integrated with the Hardhat Upgrades Library for enhanced contract deployment and initialization.

Updates

The Nexus team has mitigated the risk by employing deployment scripts, ensuring that the initialize function is invoked during the contract's deployment process.

SHB-A.13 Unbounded Loops Over Expanding Storage Arrays

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

Contracts containing loops that iterate over potentially large storage arrays can encounter scalability issues. As the array size increases, the gas cost for executing these loops might exceed the block gas limit, making functions uncallable. This issue can affect any part of the contract where loops over large, unbounded arrays are present, leading to significant performance and functionality problems.

Files Affected:

SHB-A.13.1: ValidatorExecutionRewards.sol

```
53      function updateRewardsRollup(RollupExecutionReward[] calldata
            ↪ rewards) external onlyRewardBot {
54          uint256 total_rewards;
55          for(uint i=0;i<rewards.length;i++){
56              executionRewards[rewards[i].rollupAdmin] += rewards[i].amount
                    ↪ ;
57              total_rewards+=rewards[i].amount;
58              emit RollupExecutionRewardUpdated(rewards[i].rollupAdmin,
                    ↪ rewards[i].amount);
59          }
60          if (total_rewards>(rewardsEarned-rewardsClaimed)) revert
                ↪ IncorrectRewards();
61      }
```

SHB-A.13.2: NodeOperator.sol

```
64      function addCluster(
```

```
        uint64[] calldata operatorIds,
        uint64 clusterId
    ) external onlyOwner {
        if (ssvClusters[clusterId].length != 0) revert
            ↪ ClusterAlreadyExited();
        for (uint256 i=0;i<operatorIds.length;i++){
            bytes memory ip = bytes(ssvDKGIP[operatorIds[i]]);
            if (ip.length == 0) revert OperatorNotRegistered();
        }
        ssvClusters[clusterId] = operatorIds;
        emit ClusterAdded(clusterId, operatorIds);
    }
```

SHB-A.13.3: NexusBaseBridge.sol

```
    function depositValidatorNexus(
        INexusInterface.Validator[] calldata _validators,
        uint256 stakingLimit
    ) external override onlyNexus {
        for (uint i = 0; i < _validators.length; i++) {
            bytes memory withdrawalFromCred = _validators[i]
                .withdrawalAddress[12:];
            if (
                keccak256(withdrawalFromCred) !=
                keccak256(abi.encodePacked(address(this)))
            ) revert IncorrectWithdrawalCredentials();
        }
        if (
            (((validatorCount + _validators.length) *
                (VALIDATOR_DEPOSIT) *
                BASIS_POINT) /
                (address(this).balance +
                    (validatorCount + _validators.length) *
                    (VALIDATOR_DEPOSIT))) > stakingLimit
        ) revert StakingLimitExceeding();
```

```
74          for (uint i = 0; i < _validators.length; i++) {
75              IDepositContract(DEPOSIT_CONTRACT).deposit{
76                  value: VALIDATOR_DEPOSIT
77              }(
78                  _validators[i].pubKey,
79                  _validators[i].withdrawalAddress,
80                  _validators[i].signature,
81                  _validators[i].depositRoot
82              );
83          }
84          validatorCount+=_validators.length;
85      }
```

SHB-A.13.4: Nexus.sol

```
145     function depositValidatorRollup(
146         address _rollupAdmin,
147         Validator[] calldata _validators
148     ) external override onlyOffChainBot {
149         INexusBridge(rollups[_rollupAdmin].bridgeContract)
150             .depositValidatorNexus(
151                 _validators,
152                 uint256(rollups[_rollupAdmin].stakingLimit)
153                 );
154         for (uint i = 0; i < _validators.length; i++) {
155             depositingPubkeys.addElement(_validators[i].pubKey);
156             emit ValidatorSubmitted(_validators[i].pubKey, _rollupAdmin);
157         }
158     }
```

SHB-A.13.5: Nexus.sol

```
181     function validatorExit(address rollupAdmin,bytes[] calldata pubkeys)
            ↪  external onlyOffChainBot{
182         for(uint i=0;i<pubkeys.length;i++){
```

```
183             (bool key_present, uint256 index) = activePubkeys.findElement
                    ↪ (pubkeys[i]);
184             if (key_present){
185                 activePubkeys.removeElement(pubkeys[i]);
186                 exitingKeys.addElement(pubkeys[i]);
187                 emit ValidatorExitSubmitted(rollupAdmin,pubkeys[i]);
188             }else{
189                 revert InvalidKeySupplied();
190             }
191         }
192     }
```

Recommendation:

To address this issue, it's important to limit the size of the arrays being iterated over or limit the number of iterations within a single transaction. Implementing batch processing where operations on arrays are divided into manageable chunks can effectively reduce the gas cost per transaction. Additionally, consider refactoring the code to use alternative data structures like mappings that do not require looping for access.

Updates

The Nexus team has acknowledged the issue and clarified that any for loop will not iterate for a large number, as they have a limited number of customers/partners.

SHB-A.14 Lack of Validation for SSV Operator Registration

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

Both the registerSSVOperator and updateSSVOperatorIP functions do not prevent duplicate IP addresses for SSV operators. This causes operational confusion as multiple operators could share the same IP. There is also no validation for IP address standards (IPv4 or IPv6), allowing for potential invalid IP registrations.

Files Affected:

SHB-A.14.1: NodeOperator.sol

```
40      function registerSSVOperator(uint64 _operator_id, string calldata
            ↪ _pub_key, string calldata _ip_address, string calldata name)
            ↪ external onlyOwner{
41          bytes memory ip = bytes(ssvDKGIP[_operator_id]);
42          if (ip.length != 0) revert OperatorAlreadyRegistered();
43          ssvDKGIP[_operator_id] = _ip_address;
44          emit SSVOperatorRegistered(name,_operator_id,_pub_key,_ip_address
                ↪ );
45      }
```

SHB-A.14.2: NodeOperator.sol

```
52      function updateSSVOperatorIP(uint64 _operator_id,string calldata
            ↪ _ip_address) external onlyOwner{
53          bytes memory ip = bytes(ssvDKGIP[_operator_id]);
54          if (ip.length == 0) revert OperatorNotRegistered();
55          ssvDKGIP[_operator_id] = _ip_address;
56          emit SSVOperatorUpdated(_operator_id,_ip_address);
57      }
```

Recommendation:

Implement a validation system for IP addresses in both functions:

1. IP Address Uniqueness:
    - Use a mapping to track all IP addresses.

- Check this mapping for uniqueness before registering or updating an operator's IP.
- If shared IPs are allowed, introduce additional identifiers like port numbers.

2. Standard IP Address Validation:
    - Validate IP addresses against IPv4 or IPv6 standards using regular expressions or libraries.

Updates

The Nexus team has acknowledged the issue, stating that in the current design, the validator set is permissioned, allowing the identification of operators. IP protection measures will be implemented when transitioning to a permissionless set.

SHB-A.15 Reliance on address(this).balance in getRewards

- Severity : LOW
- Likelihood : 1
- Status : Acknowledged
- Impact : 2

Description:

The getRewards function calculates rewards using address(this).balance, making the calculation vulnerable to manipulation through forced ETH transfers to the contract.

While the likelihood of such transfers is low, the use of address(this).balance introduces potential risks, as external parties can change the contract's balance, affecting reward calculations.

Files Affected:

SHB-A.15.1: NexusBaseBridge.sol

```
94      function getRewards() public view returns(uint256){
```

```
95          return (address(this).balance+(validatorCount*VALIDATOR_DEPOSIT))
                ↪  - (amountDeposited - amountWithdrawn) - slashedAmount;
96      }
```

SHB-A.15.2: NexusBridgeUserCValue.sol

```
23      function updateCValue() external {
24          uint256 rewards_to_claim = getRewards() - amountDistributed;
25          if(rewards_to_claim > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
26          uint256 _nexus_rewards = (NexusFeePercentage*rewards_to_claim)/
                ↪ BASIS_POINT;
27          amountDistributed+=rewards_to_claim-_nexus_rewards;
28          cValue = ((amountDeposited - amountWithdrawn)*CValueBasisPoint)
                ↪ /((address(this).balance-_nexus_rewards)+(validatorCount*
                ↪ VALIDATOR_DEPOSIT));
29          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
30              value: _nexus_rewards,
31              gas: 5000
32          }("");
```

Recommendation:

Replace the direct use of address(this).balance with a dedicated variable that tracks the balance relevant to reward calculations. This approach isolates the reward mechanism from unintended balance changes.

Updates

The team acknowledged the issue, stating that an additional amount of ETH sent to the contract either intentionally or by mistake won't affect the business logic of the code and can be also used as rewards.

SHB-A.16 Potential Reentrancy Attacks in Multiple Functions

- Severity : LOW
- Status : Mitigated
- Likelihood : 1
- Impact : 1

Description:

Several functions in the contract, including redeemRewards, involve external calls (call) to different addresses with a limited gas stipend. While the low gas limit reduces the likelihood of reentrancy attacks, it does not entirely eliminate the risk. External calls can potentially be exploited in reentrancy attacks, especially if the called contracts have unexpected behaviors.

Files Affected:

SHB-A.16.1: NexusBridgeDAO.sol

```
28      function redeemRewards(address reward_account) external onlyDAO {
29          uint256 total_rewards = getRewards();
30          if(total_rewards > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
31          uint256 _nexus_rewards = (NexusFeePercentage*total_rewards)/
                ↪ BASIS_POINT;
32          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
33              value: _nexus_rewards,
34              gas: 5000
35          }("");
36          if (nexus_success) {
37              emit NexusRewardsRedeemed(_nexus_rewards);
38          }
39          (bool dao_success, bytes memory dao_data) = reward_account.call{
40              value: (total_rewards - _nexus_rewards),
```

```
41              gas: 5000
42          }("");
43          rewardsClaimed += total_rewards;
44          if (dao_success) {
45              emit RewardsRedeemed((total_rewards - _nexus_rewards));
46          }
47      }
```

Recommendation:

To safeguard against reentrancy attacks, implement a reentrancy guard in functions that make external calls. This precaution is particularly crucial in functions like redeemRewards where multiple external calls are made. The steps include:

- Introduce a state variable, such as bool private inFunctionCall, to track whether a function is currently being executed.
- At the beginning of each function susceptible to reentrancy, add a check to ensure that inFunctionCall is false, and set it to true before making any external calls.
- After the external calls are completed, reset inFunctionCall to false.
- The checks would look like this:

SHB-A.16.2: NexusBridgeDAO.sol

```
require(!inFunctionCall, "ReentrancyGuard: reentrant call");
inFunctionCall = true;
// [External calls]
inFunctionCall = false;
```

Updates

The Nexus team mitigated the issue, stating that the contracts used are trusted contracts. Additionally, as a mitigation measure, the gas limit is kept low.

SHB-A.17 Potential Denial of Service (DoS) in redeemRewards

- Severity : LOW
- Likelihood : 1
- Status : Acknowledged
- Impact : 2

Description:

The redeemRewards function is vulnerable to a potential Denial of Service (DoS) attack if the reward_account is a contract that reverts in its fallback or receive function. When sending Ether to reward_account, if the contract reverts, the entire redeemRewards transaction will fail. This vulnerability can be exploited to prevent the distribution of rewards and Nexus fees.

Files Affected:

SHB-A.17.1: NexusBridgeDAO.sol

```
28      function redeemRewards(address reward_account) external onlyDAO {
29          uint256 total_rewards = getRewards();
30          if(total_rewards > VALIDATOR_DEPOSIT) revert
                ↪ WaitingForValidatorExits();
31          uint256 _nexus_rewards = (NexusFeePercentage*total_rewards)/
                ↪ BASIS_POINT;
32          (bool nexus_success, bytes memory nexus_data) = NEXUS_FEE_ADDRESS
                ↪ .call{
33              value: _nexus_rewards,
34              gas: 5000
35          }("");
36          if (nexus_success) {
37              emit NexusRewardsRedeemed(_nexus_rewards);
38          }
39          (bool dao_success, bytes memory dao_data) = reward_account.call{
```

```
40              value: (total_rewards - _nexus_rewards),
41              gas: 5000
42          }("");
```

Recommendation:

To mitigate this risk, consider implementing a more robust error handling mechanism in the redeemRewards function:

- Use a try-catch block around the call to reward_account to handle potential reverts gracefully:

SHB-A.17.2: NexusBridgeDAO.sol

```
try reward_account.call{value: (total_rewards - _nexus_rewards), gas:
    ↪ 5000}("") {
emit RewardsRedeemed((total_rewards - _nexus_rewards));
} catch {
// Handle failed transfer, e.g., log an event, attempt a retry, or take
    ↪ other remedial actions
}
```

Updates

The Nexus team acknowledged the risk, stating that the claiming address will be already known.

SHB-A.18 Missing Address Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Certain functions within the Nexus Contracts project lack address verification, allowing for the possibility of addresses being identical to address(0). This absence of address verification poses a potential security vulnerability that could lead to unintended behaviors or exploitation.

Files Affected:

SHB-A.18.1: Nexus.sol

```
76      function setOffChainBot(address _botAddress) external onlyOwner {
77          offChainBot = _botAddress;
78      }
```

SHB-A.18.2: Nexus.sol

```
84      function setNodeOperatorContract(address _nodeOperator) external
            ↪ onlyOwner{
85          NodeOperatorContract=_nodeOperator;
86          emit NodeOperatorContractChanged(_nodeOperator);
87      }
```

SHB-A.18.3: Nexus.sol

```
89      function changeExecutionFeeAddress(address _execution_fee_address)
            ↪ external onlyOwner {
90          ISSVNetworkCore(SSV_NETWORK).setFeeRecipientAddress(
                ↪ _execution_fee_address);
91      }
```

SHB-A.18.4: ValidatorExecutionRewards.sol

```
33      constructor(address _rewardBot){
34          rewardBot = _rewardBot;
35          emit ChangeRewardBotAddress(_rewardBot);
```

SHB-A.18.5: ValidatorExecutionRewards.sol

```
39      function changeRewardBotAddress(address _bot_address) external
            ↪ onlyOwner{
40          rewardBot = _bot_address;
41          emit ChangeRewardBotAddress(_bot_address);
42      }
```

SHB-A.18.6: NexusOwnable.sol

```
39      function transferOwnership(address newOwner) external onlyOwner{
40          emit OwnerChanged(owner, newOwner);
41          owner = newOwner;
42      }
```

Recommendation:

To address this issue, implement robust address verification checks in the relevant functions of the Nexus Contracts project. Ensure that the provided addresses are distinct from address(0) to enhance security and prevent potential misuse or vulnerabilities.

Updates

The Nexus team resolved the issue by implementing our recommendation and incorporating zero address checks.

SHB-A.19 Floating Pragma

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

All the contracts use a floating Solidity pragma of 0.8.19, indicating that they can be compiled with any compiler version from 0.8.19 (inclusive) up to, but not including, version

0.9.0.This flexibility could potentially introduce unexpected behavior if the contracts are compiled with a newer compiler version that includes breaking changes.

Files Affected:

SHB-A.19.1: Nexus.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.2: NexusBaseBridge.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.3: NexusBridgeDAO.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.4: NexusBridgeUserCValue.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.5: NexusBridgeUserRebase.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.6: NexusDAIBridge.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.7: nexusLibrary.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.8: NexusOwnable.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.9: NexusProxy.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.10: UUPSUpgreadable.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.11: NodeOperator.sol

```
2 pragma solidity ^0.8.19;
```

SHB-A.19.12: ValidatorExecutionRewards.sol

```
2 pragma solidity ^0.8.19;
```

Recommendation:

It is generally recommended to lock the pragma statement to a specific Solidity compiler version to ensure consistent behavior across different compiler versions. To achieve this, consider removing the caret (^) from the pragma statement and specifying a fixed version, such as pragma solidity 0.8.19;.

Updates

The Nexus team has resolved this issue by fixing the pragma version across all Nexus contracts, locking it to 0.8.19.

SHB-A.20 Inconsistent NEXUS_NETWORK Address in Nexus Contracts

- Severity : INFORMATIONAL
- Likelihood : 1
- Status : Fixed
- Impact : 0

Description:

The NexusLibrary and NexusBaseBridge contracts currently use different NEXUS_NETWORK addresses. This inconsistency can impact the logic of the Nexus

contract, introducing potential issues in the interaction between these components. Even for testing purposes, maintaining a consistent NEXUS_NETWORK address is crucial to ensure accurate and reliable contract behavior.

Files Affected:

SHB-A.20.1: nexusLibrary..sol

```
14  address public constant NEXUS_NETWORK =
15      0x7610dd2DE44aA3c03313b4c2812C482D86F3a9e7;
```

SHB-A.20.2: NexusBaseBridge.sol

```
19      address public override NEXUS_NETWORK = 0
            ↪ xd1C788Ac548Cb467b3c4B14CF1793BCa3c1dCBEB;
```

Recommendation:

Ensure that the NEXUS_NETWORK address is consistent across the Nexus contracts, maintaining alignment for both production and testing environments.

Updates

The Nexus team has addressed the issue by modifying the NEXUS_NETWORK address in both NexusLibrary and NexusBaseBridge contracts to be consistent.

B Off-chain

SHB-B.1 Exposing Private Key Through Command Line Arguments

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The KeyGeneration class is instantiated with a command line argument that includes a private key (nexus = KeyGeneration(args.config, args.private_key, args.dkg_enabled)). Passing sensitive information like a private key via command line arguments poses a security risk. Command line arguments are often easily accessible through various means, such as the command history of the terminal, process listing, or server logs. This exposure could lead to unauthorized access to the private key.

Files Affected:

SHB-B.1.1: create_keys.py

```
352     nexus = KeyGeneration(args.config, args.private_key, args.
            ↪ dkg_enabled)
```

Recommendation:

To enhance security, consider alternative methods of supplying the private key to the script:

- Environment Variables: Use an environment variable to pass the private key. Environment variables can be set at the system level or within a secure application context and accessed by the script, for example:

SHB-B.1.2: create_keys.py

```
import os
private_key = os.environ.get("PRIVATE_KEY")
nexus = KeyGeneration(args.config, private_key, args.dkg_enabled)
```

- Secure Configuration File: Store the private key in a secure, access-controlled configuration file. Read the key from this file within your script, and ensure file permissions are set to restrict access to authorized users only.
- Prompt for Input: Prompt the user to enter the private key during runtime. This method keeps the key out of the terminal history and server logs, for example:

SHB-B.1.3: create_keys.py

```
import getpass
private_key = getpass.getpass("Enter private key: ")
nexus = KeyGeneration(args.config, private_key, args.dkg_enabled)
```

- Key Management Services: Utilize a key management service or a secrets manager to handle the private key securely. These services offer heightened security measures and access control.

Updates

The Nexus team has fixed the issue by using the PwdAction class that call the getpass function.

SHB-B.2 Lack of Deposit Transaction Status Verification

- Severity : HIGH
- Status : Fixed
- Likelihood : 3
- Impact : 2

Description:

The submit_transaction method in create_keys.py file submits a deposit transaction without checking the deposit_status value of the transaction after its execution. This introduces a potential risk, as the success or failure of the deposit transaction is crucial for maintaining accurate state information.

Files Affected:

SHB-B.2.1: create_keys.py

```
220     def submit_transaction(self, rollup,operators):
221         logging.info("submitting deposit transaction")
222         gp_transaction = int(self.eth_node.eth_node.eth.gas_price * 1.5)
```

```
223         rollup_transactions = self.state[rollup.ownerAddress]["
                ↪ latest_validator_keys"]
224         deposit_tx = rollup_transactions["deposit"]["transaction"]
225         if rollup_transactions["deposit"]["transaction_hash"] == "":
226             deposit_tx["gasPrice"] = gp_transaction
227             deposit_status, tx_hash = self.eth_node.make_tx(deposit_tx)
228             rollup_transactions["deposit"]["transaction_hash"] = tx_hash
```

Recommendation:

It is recommended to enhance the submit_transaction method by including a check for the deposit transaction status (deposit_status). Ensure that the transaction is successfully executed before updating the state with the transaction hash. This verification step helps prevent inaccurate state information and ensures the reliability of the deposit process.

Updates

The Nexus team has fixed the issue by raising an exception RuntimeError("Key deposit failed") if the deposit_status is False.

SHB-B.3 Incomplete Implementation of Functions in Off-chain Code

- Severity : HIGH
- Likelihood : 3
- Status : Acknowledged
- Impact : 2

Description:

The off-chain code source contains three functions, namely exit_dkg, generate_exit, and get_validators, which are marked as incomplete or to-do. The absence of these implementations may impact the intended logic or functionality of the system. exit_dkg and generate_exit are not implemented, and get_validators does not return any value.

Files Affected:

SHB-B.3.1: create_keys.py

```
327     def exit_dkg(self, rollup, validator_count):
328         """
330         :param rollup:
331         :param validator_count:
332         :return:
333         """
334         logging.info("Exiting keys for rolluo")
```

SHB-B.3.2: ssv_dkg.py

```
51     def generate_exit(self):
52         """
53         todo: implement this for when ssv enables exits
54         :return:
55         """
```

SHB-B.3.3: subgraph.py

```
48     def get_validators(self):
49         query = """{
50             rollups(
51                 first: 1000
52             ) {
53                 id
54                 }
55             }
56         """
```

Recommendation:

Consider implementing each of the mentioned functions, exit_dkg, generate_exit, and get_validators, to ensure accurate handling of the business logic and return the expected

values. This implementation is crucial for the proper functionality of the off-chain code and to achieve the intended system behavior.

Updates

The Nexus team has acknowledged the risk, and they mentioned that will implement those function once SSV implements withdrawals using DKG.

SHB-B.4 Potential File Content Extraction

- Severity : MEDIUM
- Likelihood : 1
- Status : Acknowledged
- Impact : 3

Description:

The bot design allows for reading data from a file path specified in the state.json file under the deposit_file attribute. If an attacker gains the ability to modify state.json, they can manipulate the deposit_file attribute to point to any JSON file on the system. When the application reads this file, sensitive data could be inadvertently exposed. Although the resulting transaction may fail, the contents of the manipulated file could be revealed through transaction analysis.

Files Affected:

SHB-B.4.1: create_keys.py

```
298         for key in self.state["dkg"]:
299             if not self.state["dkg"][key]["status_deposit"]:
300                 deposit_data = Helpers.read_file(self.state["dkg"][key]["
                        ↪ deposit_file"], as_dict=True)
301                 validator = Validator("0x" + deposit_data[0]["pubkey"],
302                                       "0x" + deposit_data[0]["
                                              ↪ withdrawal_credentials"],
```

```
303                                     "0x" + deposit_data[0]["signature"],
304                                     "0x" + deposit_data[0]["
                                           ↪ deposit_data_root"])
305                 tx = self.nexus_contract.submit_validator(rollup.
                        ↪ ownerAddress, [validator])
```

Recommendation:

To mitigate this security risk, implement stringent file path validation and access controls:

- Validate File Paths: Introduce checks to ensure that file paths in the state.json file adhere to expected patterns or are within specific directories. This can prevent arbitrary file paths from being processed. For example:

SHB-B.4.2: create_keys.py

```
allowed_path_prefix = "/expected/directory/"
if not deposit_file_path.startswith(allowed_path_prefix):
raise SecurityException("Invalid file path")
```

- Restrict File Access: Implement access controls to restrict which files the application can read. This can be done at the operating system level or within the application.
- Secure state.json: Ensure that state.json is stored securely with appropriate permissions to prevent unauthorized modifications.

Updates

The Nexus team acknowledged the risk, stating that they will put permissions on state.json. This will be achieved through user access, allowing only the script to write to the state.json. Consequently, only the script will have the ability to modify it.

SHB-B.5 Handling Failed Transactions in Automated Bot Process

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

In the current implementation, if a transaction fails (e.g., status_tx is False), the bot does not perform any error handling or retry mechanism. This lack of response to a failed transaction can lead to issues where necessary actions are not completed, potentially causing synchronization problems or other operational inconsistencies.

Files Affected:

SHB-B.5.1: create_keys.py

```
305                 tx = self.nexus_contract.submit_validator(rollup.
                        ↪ ownerAddress, [validator])
306                 tx["gasPrice"] = gp_transaction
307                 status_tx, tx_hash = self.eth_node.make_tx(tx)
308                 if status_tx:
309                     self.state["dkg"]["0x" + deposit_data[0]["pubkey"]]["
                            ↪ status_deposit"] = True
```

Recommendation:

Implement a robust error handling and retry mechanism for transaction failures to ensure the bot can recover and complete its intended actions:

- Error Handling: When status_tx is False, log the failure and the reason for the transaction failure, if available, and Implement a mechanism to either retry the transaction after a certain interval or mark it for manual review, depending on the nature of the failure.

- Retry Mechanism: Introduce a retry limit to avoid infinite retry loops, and implement exponential backoff or a fixed delay between retries to manage load and give time for potential issues to resolve.
- State Management: Maintain the state of each transaction attempt. If a transaction fails and is retried, ensure that the state reflects these attempts accurately, and consider storing failed transaction details for further analysis and resolution.
- Notification System: Introduce a notification system to alert administrators or relevant parties in case of persistent transaction failures.

Updates

The team has fixed the issue by raising the exception RuntimeError("key deposit failed") if the transaction failed.

SHB-B.6 Inaccurate Local RPC Check in EthNode Class

- Severity : LOW
- Status : Acknowledged
- Likelihood : 2
- Impact : 1

Description:

The EthNode class contains an inaccurate check to determine if the RPC URL points to a local node. The current implementation checks if the string '127.0.0.1' or 'localhost' exists in the rpc_url. However, due to the way the conditional statement is written, any non-empty string will satisfy the first condition, leading to false positives. For example, an URL like 'localhostaaa' will incorrectly be identified as a local RPC URL.

Files Affected:

SHB-B.6.1: ethereum_connector.py

```
14      def __init__(self, rpc_url, private_key):
```

```
15          self.eth_node = Web3(Web3.HTTPProvider(rpc_url))
16          # self.eth_node.eth.set_gas_price_strategy(rpc_gas_price_strategy
                ↪ )
17          if '127.0.0.1' or 'localhost' in rpc_url:
18              # w3.eth.accounts()[0]
19              self.local = True
```

Recommendation:

Modify the if-statement to correctly check for both 127.0.0.1 and localhost in the rpc_url.

Updates

The Nexus team acknowledged the risk since the setup will be done by the Nexus team.

SHB-B.7 Potential Race Condition in get_node_operators Method

- Severity : LOW
- Likelihood : 1
- Status : Acknowledged
- Impact : 2

Description:

The get_node_operators method fetches data through a GraphQL query, executed at a one-minute interval. This frequency of calls poses a risk of race conditions, where the data could change in the period between subsequent calls. If the data is used in operations dependent on its accuracy at a specific point in time, any change during or after the call could lead to inconsistencies or errors in processing.

Files Affected:

SHB-B.7.1: subgraph.py

```
58      def get_node_operators(self, operator_ids):
59          query = """{
60            nodeOperators(where:{id_in:operator_ids}){
61                  id
62                  name
63                  ip
64                  pubkey
65                  }
66              }
67          """.replace("operator_ids",json.dumps(operator_ids))
68          result = self.make_call(query)["data"]["nodeOperators"]
69          nodeOperators = []
70          for operator in result:
71              nodeOperators.append({"id":int(operator["id"]),"ip": operator
                ↪ ["ip"],"public_key":operator["pubkey"]})
72          return nodeOperators
```

Recommendation:

Consider conducting a formal evaluation to determine if the current one-minute frequency of fetching operator data is suitable for the bot's purposes. If deemed excessive, it is advisable to reduce the frequency to optimize operational efficiency

Updates

The Nexus team acknowledged the issue, stating that a frequency of 1mn is good, they are planning to increase it when decentralizing the bot.

SHB-B.8 Inefficiencies and Misnomers in get_latest_nonce Function

- Severity : UNDETERMINED
- Status : Acknowledged
- Likelihood : 0
- Impact : 0

Description:

The get_latest_nonce function contains several issues impacting its accuracy, efficiency, and naming convention:

- Misleading Naming: The function's name suggests it retrieves the latest nonce, but in reality, it counts the number of ValidatorAdded events. This discrepancy can lead to confusion.
- Outdated Starting Block: The loop's starting point (to_block > 9849964) uses a significantly old block number, potentially leading to unnecessary iterations and performance inefficiencies.
- Variable Naming Confusion: The variable result is used to capture the output of filter_deploy.get_all_entries(), but subsequent checks and increments mistakenly refer to a non-existent results variable.
- Unreachable Code: The check if len(results) > 0 will never be true because results remains an empty dictionary, making this part of the code redundant.
- Lack of Caching: The function can benefit from caching, especially if the number of ValidatorAdded events for a specific owner does not change frequently.

Files Affected:

SHB-B.8.1: ssv_utils.py

```
85      def get_latest_nonce(self, owner_address):
86          step = 10000
87          from_block = self.web3.eth.get_block_number() - step
```

```
        to_block = self.web3.eth.get_block_number()
        nonce = 0
        while to_block > 9849964:
            filter = self.contract.events.ValidatorAdded.build_filter()
            results = {}
            filter.fromBlock = from_block
            filter.toBlock = to_block
            filter.args.owner.match_single(owner_address)
            filter_deploy = filter.deploy(self.web3)
            result = filter_deploy.get_all_entries()
            if len(result) > 0:
                nonce += len(result)
            print(results)
            if len(results) > 0:
                return results[max(results)]
            to_block = from_block
            from_block -= step
        return nonce
```

Updates

The Nexus team has acknowledged the issue since it's analogous to the SSV system.

4 Best Practices

A Nexus Contracts

BP-A.1 Eliminate Unused Import for Withdrawal Contract

Description:

The Nexus Contract currently incorporates an unused import statement for the Withdrawal contract. Unused imports contribute to code clutter, potentially impacting readability and increasing the risk of overlooking critical code elements. It is considered a best practice to regularly review and remove any unused import statements to maintain a concise and organized codebase.

Files Affected:

BP-A.1.1: Nexus.sol

```
4 import {Withdraw} from "./Withdrawal.sol";
```

Status - Fixed

BP-A.2 Utilize Mapping for Validator Public Keys Status Instead of Separate Arrays

Description:

The current Nexus Contract employs distinct arrays, namely depositingPubkeys, activePubkeys, and exitingKeys, to manage validators public keys based on their status. A more efficient and gas-effective approach involves implementing a mapping structure that associates each validator's public key with its status (represented by an enumeration [DEPOSIT, ACTIVE, EXIT, ...]). This approach provides a more organized and efficient way to manage validator status, improving code clarity and potentially reducing gas costs associated with array manipulations.

Files Affected:

BP-A.2.1: Nexus.sol

```
34      bytes[] public depositingPubkeys;
35      bytes[] public activePubkeys;
36      bytes[] public exitingKeys;
```

Status - Fixed

BP-A.3 Optimize Gas Consumption and Enhance Readability In depositValidatorNexus

Description:

In the depositValidatorNexus function within the NexusLibrary and NexusBaseBridge contracts, consider creating a local variable to calculate the product of (validatorCount + _validators.length) * (VALIDATOR_DEPOSIT) for improved gas consumption and enhanced code readability. The function can then utilize this local variable to simplify calculations.

Files Affected:

BP-A.3.1: nexusLibrary.sol

```
103         if (
104             (((validatorCount + _validators.length) *
105                 (VALIDATOR_DEPOSIT) *
106                 BASIS_POINT) /
107                 (address(this).balance +
108                     (validatorCount + _validators.length) *
109                     (VALIDATOR_DEPOSIT))) > stakingLimit
110         ) revert StakingLimitExceeding();
```

BP-A.3.2: NexusBaseBridge.sol

```
65          if (
66              (((validatorCount + _validators.length) *
```

```
                (VALIDATOR_DEPOSIT) *
                BASIS_POINT) /
                (address(this).balance +
                    (validatorCount + _validators.length) *
                    (VALIDATOR_DEPOSIT))) > stakingLimit
        ) revert StakingLimitExceeding();
```

Status - Fixed

BP-A.4 Combine Loop Iterations for Improved Readability

Description:

In the depositValidatorNexus function, consider consolidating the first and last for loops into a single loop to iterate over the _validators array once. Perform both withdrawal credentials validation and deposit actions within this consolidated loop. Combining the loops eliminates duplicated code and simplifies the overall structure of the function.

Files Affected:

BP-A.4.1: NexusBaseBridge.sol

```
    function depositValidatorNexus(
        INexusInterface.Validator[] calldata _validators,
        uint256 stakingLimit
    ) external override onlyNexus {
        for (uint i = 0; i < _validators.length; i++) {
            bytes memory withdrawalFromCred = _validators[i]
                .withdrawalAddress[12:];
            if (
                keccak256(withdrawalFromCred) !=
                keccak256(abi.encodePacked(address(this)))
            ) revert IncorrectWithdrawalCredentials();
        }
        if (
```

```
            (((validatorCount + _validators.length) *
                (VALIDATOR_DEPOSIT) *
                BASIS_POINT) /
                (address(this).balance +
                    (validatorCount + _validators.length) *
                    (VALIDATOR_DEPOSIT))) > stakingLimit
        ) revert StakingLimitExceeding();

        for (uint i = 0; i < _validators.length; i++) {
            IDepositContract(DEPOSIT_CONTRACT).deposit{
                value: VALIDATOR_DEPOSIT
            }(
                _validators[i].pubKey,
                _validators[i].withdrawalAddress,
                _validators[i].signature,
                _validators[i].depositRoot
            );
        }
        validatorCount+=_validators.length;
    }
```

BP-A.4.2: nexusLibrary.sol

```
    function depositValidatorNexus(
        INexusInterface.Validator[] calldata _validators,
        uint256 stakingLimit
    ) external onlyNexus {
        for (uint i = 0; i < _validators.length; i++) {
            bytes memory withdrawalFromCred = _validators[i]
                .withdrawalAddress[12:];
            if (
                keccak256(withdrawalFromCred) !=
                keccak256(abi.encodePacked(address(this)))
            ) revert IncorrectWithdrawalCredentials();
        }
```

```
102         uint256 validatorCount = getVariable(VALIDATOR_COUNT_SLOT);
103         if (
104             (((validatorCount + _validators.length) *
105                 (VALIDATOR_DEPOSIT) *
106                 BASIS_POINT) /
107                 (address(this).balance +
108                     (validatorCount + _validators.length) *
109                     (VALIDATOR_DEPOSIT))) > stakingLimit
110         ) revert StakingLimitExceeding();
111 
112         for (uint i = 0; i < _validators.length; i++) {
113             IDepositContract(DEPOSIT_CONTRACT).deposit{
114                 value: VALIDATOR_DEPOSIT
115             }(
116                 _validators[i].pubKey,
117                 _validators[i].withdrawalAddress,
118                 _validators[i].signature,
119                 _validators[i].depositRoot
120             );
121         }
122         validatorCount += _validators.length;
123     }
```

Status - Fixed

BP-A.5 Rename Proxiable Contract File for Clarity

Description:

In the Nexus Contracts project, it is advised to rename the file currently named UUPSUpgreadable.sol to better align with the contained contract named Proxiable. This recommended change aims to enhance clarity and maintain consistency within the project. Consider choosing a name that accurately represents the purpose of the contract, facilitating easier understanding for developers and minimizing potential naming conflicts.

Files Affected:

BP-A.5.1: UUPSUpgreadable.sol

```
4  /**
5   * @title Proxiable
6   * @author RohitAudit
7   * @dev This contract is implemented in the implementation contract to
       ↪ make it upgreadable. Removing this contract
8   * will remove the upgreadability feature of the contract
9   */
10 contract Proxiable {
```

Status - Fixed

BP-A.6 Utilize Constants for Code Positioning In Proxiable Contract

Description:

In the Proxiable contract, enhance code readability and maintainability by utilizing a constant value for the code position in storage. Replace the hard-coded value 0xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7 with a named constant. Declare this constant globally in the contract to represent the keccak256 hash of PROXIABLE (bytes32 constant PROXIABLE_CODE_POSITION = keccak256("PROXIABLE");).

Files Affected:

BP-A.6.1: UUPSUpgreadable.sol

```
14     function updateCodeAddress(address newAddress) internal {
15 if(bytes32(0
       ↪ xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7
       ↪ ) != Proxiable(newAddress).proxiableUUID()) revert NotCompatible
       ↪ ();
16         assembly { // solium-disable-line
```

```
17  sstore(0
        ↪ xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7
        ↪ , newAddress)
18          }
19      }
20      function proxiableUUID() public pure returns (bytes32) {
21  return 0
        ↪ xc5f16f0fcc639fa48a6947836d9850f504798523bf8c9a3a87d5876cf622bcf7
        ↪ ;
22      }
```

Status - Fixed

BP-A.7 Remove Direct Initialization in Nexus Contract

Description:

Remove the direct initialization of variables in the Nexus contract, considering the contract's upgradability. Initialization values directly set in the contract declaration may not be set during implementation upgrades. Instead, utilize the initialize function for proper initialization and persistence in the upgraded implementation.

Files Affected:

BP-A.7.1: Nexus.sol

```
32      address public offChainBot = 0
            ↪ x45a3f77543167c8D0965194879c4e0B0dbB581d0;
33      mapping(address => Rollup) public rollups;
34      bytes[] public depositingPubkeys;
35      bytes[] public activePubkeys;
36      bytes[] public exitingKeys;
37      mapping(uint256=>uint16) polygonCDKPartners;
38      address public NodeOperatorContract;
39
```

```
40      // change these addresses to mainnet address when deploying on
            ↪ mainnet
41      address private constant SSV_NETWORK =
42          0xC3CD9A0aE89Fff83b71b58b6512D43F8a41f363D;
43      address private constant SSV_TOKEN =
44          0x3a9f01091C446bdE031E39ea8354647AFef091E7;
45      uint16 private constant BASIS_POINT = 10000;
```

Status - Acknowledged

B Off-chain

BP-B.1 Make Private Key Argument Required in Command Line Parser

Description:

The command line parser for the private key argument in the Nexus bot includes the help parameter but lacks the required=True specification. Making the private key argument required ensures that users explicitly provide the private key when executing the command, preventing accidental omissions and potential issues during bot operation.

Files Affected:

BP-B.1.1: create_keys.py

```
337 if __name__ == '__main__':
338     parser = argparse.ArgumentParser(description="This script is used
            ↪ for creating validator keys for Nexus")
339     parser.add_argument("-c", "--config", help="config file to run the
            ↪ script. Refer README.md", required=True)
340     parser.add_argument("-priv", "--private-key", help="private key for
            ↪ whitelisted nexus bot")
341     parser.add_argument('-v', '--verbose', help="Verbose Logging",
            ↪ action="store_const", dest="loglevel",
```

Status - Fixed

BP-B.2 Optimization In get_rollups

Description:

The use of an if-else block in get_rollups function to check and assign default values leads to code redundancy and potential inefficiency. Replace the if-else block with the .get() method. This method allows setting a default value if a specified key is not found.

BP-B.2.1: subgraph.py

```
rollups = []
for rollup in self.make_call(query)["data"]["rollups"]:
validator_count = rollup_data.get("validatorCount", 0)
rollups.append(
Rollup(Web3.toChecksumAddress(rollup["id"]), Web3.toChecksumAddress(
    ↪ rollup["bridgeContract"]),
rollup["stakingLimit"], validator_count , rollup["clusterId"]))
return rollups
```

Files Affected:

BP-B.2.2: subgraph.py

```
37          for rollup in self.make_call(query)["data"]["rollups"]:
38              if rollup["validatorCount"] is None:
39                  rollups.append(
40                      Rollup(Web3.toChecksumAddress(rollup["id"]), Web3.
                            ↪ toChecksumAddress(rollup["bridgeContract"]),
41                             rollup["stakingLimit"], 0, rollup["clusterId"]))
42              else:
43                  rollups.append(
44                      Rollup(Web3.toChecksumAddress(rollup["id"]), Web3.
                            ↪ toChecksumAddress(rollup["bridgeContract"]),
45                             rollup["stakingLimit"], rollup["validatorCount
                                ↪ "], rollup["clusterId"]))
```

```
46          return rollups
```

Status - Acknowledged

BP-B.3 Implementing Caching Mechanism for Rollup Data

Description:

The current implementation involves repeatedly fetching rollup data every minute with self.subgraph.get_rollups(). This approach can be inefficient, especially if the rollup data does not change frequently. Constantly sending the same request can lead to unnecessary load on the server and can potentially create synchronization issues if the data changes during the request interval. Introduce a caching mechanism to store and reuse rollup data, reducing the frequency of requests:

- Implement a cache for storing rollup data. This cache can be a simple in-memory store or a more sophisticated caching solution, depending on the application's scale and complexity.
- Update the logic to check the cache first before fetching new data:
- On the first iteration, fetch the rollup data and store it in the cache.
- In subsequent iterations, check if the data in the cache is still valid (e.g., not older than a certain threshold). If it is, use the cached data; otherwise, fetch new data and update the cache.
- Consider adding a configurable time-to-live (TTL) for the cached data. This TTL should be based on how frequently the rollup data changes.

Files Affected:

BP-B.3.1: create_keys.py

```
48      def start(self):
49          logging.info("starting the script")
50          try:
51              while True:
```

```
52                 logging.info("getting rollups")
53                 rollups = self.subgraph.get_rollups()
```

Status - Acknowledged

BP-B.4 Using a JSON Mapping for Chain IDs and Network Names

Description:

The current implementation uses hard-coded conditionals to map Ethereum chain IDs to network names. This approach is not scalable or easily maintainable, especially if there are frequent updates or additions to the supported networks. Changes in network configurations would require altering the codebase, leading to potential errors and requiring redeployment. Switch to a JSON-based mapping system for chain IDs and network names. This approach allows for easier updates and additions without needing to modify the code:

- Create a JSON file (e.g., network_mappings.json) containing mappings between chain IDs and their corresponding network names. Structure it as follows:

BP-B.4.1: network_mappings.json

```
{
"5": "prater",
"1": "mainnet",
// Add more mappings as needed
}
```

- Load this JSON file into your application and use it to look up network names based on the chain ID:

BP-B.4.2: create_keys.py

```
import json
# Load network mappings
with open('network_mappings.json') as f:
```

```
network_mappings = json.load(f)
chain_id = str(self.eth_node.eth_node.eth.chainId)
if chain_id in network_mappings:
network_name = network_mappings[chain\_id]
logging.info(f"Starting DKG process for {network_name}")
dkg = DKG(network_name)
else:
logging.error("Unsupported chain ID")
```

Files Affected:

BP-B.4.3: create_keys.py

```
59                  if self.is_dkg_enabled:
60                      if self.eth_node.eth_node.eth.chainId == 5:
61                          logging.info("Starting DKG process for goerli")
62                          dkg = DKG("prater")
63                      elif self.eth_node.eth_node.eth.chainId == 1:
64                          logging.info("Starting DKG process for mainnet
                                ↪ ")
65                          dkg = DKG("mainnet")
66                      else:
67                          raise ValueError("Wrong network detected")
```

Status - Acknowledged

BP-B.5 Reducing Iterations in get_latest_cluster Function

Description:

The get_latest_cluster function iterates through blocks and events to find a matching transaction, but continues to iterate even after finding a relevant match. Since the search starts from the to_block (the most recent block) and moves backwards, the first matching transaction found is the most recent one.

Continuing the search beyond this point is unnecessary and inefficient. Optimize the function by stopping the iteration and returning the result as soon as the first matching transaction is found:

- Early Return Upon Match: Modify the loop to return the result immediately when a matching transaction is found. This can be done by checking the condition after processing each filter and breaking out of the loop if a match is found.
- Optimize Block Range: Adjust the block range (from_block and to_block) if there are ways to estimate a more accurate starting point for the search, reducing the number of blocks to iterate over.
- Limit Number of Iterations: Implement a limit on the number of iterations or blocks to search backward, especially if there's a reasonable expectation of how far back a relevant transaction might be.

Files Affected:

BP-B.5.1: ssv_utils.py

```
55      def get_latest_cluster(self, owner_address, operator_ids):
56          step = 10000
57          from_block = self.web3.eth.get_block_number() - step
58          to_block = self.web3.eth.get_block_number()
59          print(to_block)
60          while to_block > 9849964:
61              filters = [self.contract.events.ValidatorAdded.build_filter()
                    ↪ ,
62                         self.contract.events.ValidatorRemoved.build_filter()
                           ↪ ,
63                         self.contract.events.ClusterLiquidated.build_filter
                           ↪ (),
64                         self.contract.events.ClusterReactivated.build_filter
                           ↪ (),
65                         self.contract.events.ClusterWithdrawn.build_filter()
                           ↪ ,
```

```
66                      self.contract.events.ClusterDeposited.build_filter()
                            ↪ ]
67              results = {}
68              for filter_result in filters:
69                  filter_result.fromBlock = from_block
70                  filter_result.toBlock = to_block
71                  filter_result.args.owner.match_single(owner_address)
72                  filter_deploy = filter_result.deploy(self.web3)
73                  result = filter_deploy.get_all_entries()
74                  if len(result) > 0:
75                      for data in result:
76                          if sorted(result[0].args.operatorIds) == sorted(
                                ↪ operator_ids):
77                              results[data.blockNumber] = data.args.cluster
78              print(results)
79              if len(results) > 0:
80                  return results[max(results)]
81              to_block = from_block
82              from_block -= step
83          return [0, 0, 0, True, 0]
```

Status - Acknowledged

BP-B.6 Duplicate Field in GraphQL Query within get_cluster Method

Description:

The get_cluster method contains a GraphQL query with a duplicate field. Specifically, operatorIds appears twice in the query. This duplication is unnecessary and may lead to confusion or potential issues with query processing. Remove the duplicate field from the GraphQL query to streamline and clarify the query. The updated query should only list each required field once.

Files Affected:

BP-B.6.1: subgraph.py

```
74      def get_cluster(self, cluster_id):
75          query = """ {
76               clusters(where: {id: _given_id}) {
77                 operatorIds
78                 id
79                 operatorIds
80                 }
81             }
82           """.replace("_given_id", json.dumps(cluster_id))
83          return self.make_call(query)["data"]["clusters"][0]["operatorIds
                ↪ "]
```

Status - Fixed

BP-B.7 Remove Unnecessary Operator Fee Calculation in start_dkg_ceremony

Description:

The start_dkg_ceremony method contains a loop for calculating the total operator fee for each operator in the specified cluster (operator_fee_total). However, the calculated total is neither used nor returned to the current implementation. This redundant calculation may consume resources without providing any meaningful outcome. It is recommended to remove the loop responsible for calculating operator_fee_total in the start_dkg_ceremony method, as it neither influences the method's logic nor contributes to its final result. By eliminating this unnecessary computation, the efficiency of the method can be improved.

Files Affected:

BP-B.7.1: create_keys.py

```
273         operator_fee_total = 0
```

```
274         for operator in operator_ids:
275             operator_fee_total += ssv_network_views.get_operator_fee(int(
                ↪ operator))
276         for validator in range(validator_count):
277             deposit_file, keyshare = dkg.generate_validator(nonce, ",".
                ↪ join(operator_ids), operator_info,
278                                                             rollup.
                                                                ↪ bridgeContract
                                                                ↪ , self.config
                                                                ↪ .contracts.
                                                                ↪ nexus.address
                                                                ↪ ,
279                                                             rollup.
                                                                ↪ bridgeContract
                                                                ↪ )
280             deposit_data = Helpers.read_file(deposit_file, as_dict=True)
281             self.state["dkg"] = {
282                 "0x" + deposit_data[0]["pubkey"]: {"deposit_file":
                    ↪ deposit_file, "status_deposit": False,
283                                                    "keyshare_file": keyshare,
                                                       ↪ "status_share": False
                                                       ↪ }}
284             nonce += 1
```

Status - Fixed

BP-B.8 Remove the unused import statements

Description:

The create_keys.py file includes an unused import for Web3 from the web3 library, while the helpers.py file includes an unused import for ValidatorShares from the utils.types module. Eliminating these unused imports contributes to a cleaner and more efficient codebase and minimizes unnecessary dependencies.

Files Affected:

BP-B.8.1: aws.py

```
1 from web3 import Web3
```

BP-B.8.2: helpers.py

```
6 from utils.types import ValidatorShares
```

Status - Fixed

BP-B.9 Eliminate Dead Code in Off-chain Files

Description:

Review and remove any dead or unused code snippets present in the off-chain files. Dead code, which includes commented-out sections, can clutter the codebase and make it harder to understand. Removing these unnecessary elements enhances code readability, reduces confusion, and ensures a more maintainable project.

Files Affected:

BP-B.9.1: create_keys.py

```
98             # self.alerting.send_alert(traceback.format_exc())
```

BP-B.9.2: create_keys.py

```
200         # initial_nonce = 0
```

BP-B.9.3: ethereum_connector.py

```
5 # from web3.gas_strategies.rpc import rpc_gas_price_strategy
```

BP-B.9.4: ethereum_connector.py

```
16         # self.eth_node.eth.set_gas_price_strategy(rpc_gas_price_strategy
               ↪ )
17         if '127.0.0.1' or 'localhost' in rpc_url:
18             # w3.eth.accounts()[0]
```

BP-B.9.5: ethereum_connector.py

```
27          # self.eth_node.eth.call(tx)
28          tx['nonce'] = self.eth_node.eth.get_transaction_count(
29              self.account.address)
30          # if self.local:
31          # tx.pop('maxFeePerGas')
```

BP-B.9.6: helpers.py

```
58          # print(asyncio.run(self.bot.get_me()))
```

BP-B.9.7: ssv_cli.py

```
10  # output = check_output(["./ssv-cli", "key-shares", "-ks",
11  # "/home/rohit/Documents/hackathon-bogota/ssv-service/validator_keys/
        ↪ keystore-m_12381_3600_1_0_0-1665236798.json",
12  # "-ps", "test", "-oid", "1,2,9,42", "-ok",
```

BP-B.9.8: ssv_cli.py

```
48          # print("operator_data")
49          # print(operator_data)
```

Status - Acknowledged

5 Tests

Results:

→ DAO bridge test

✓ should implement slashing (637ms)

✓ should be able to claim his DAO rewards (602ms)

✓ should be able to update exited validators (591ms)

→ Rebase bridge contract for rollup

✓ should implement slashing (194ms)

✓ should rebase tokens (99ms)

✓ should be able to update exited validators (718ms)

→ CValue bridge contract for rollup

✓ should implement slashing (242ms)

✓ should change c-token value (182ms)

✓ should be able to update exited validators (408ms)

→ Execution Reward test

✓ should receive execution rewards (121ms)

✓ bot should update execution rewards for the rollup (340ms)

✓ rollupadmin should be able to claim rewards (228ms)

→ nexus test

✓ should send SSV to nexus Contract (527ms)

✓ should whitelist rollup (328ms)

✓ should register rollup (1228ms)

✓ should change staking limit (156ms)

✓ should change nexus fee limit (328ms)

✓ should change cluster (394ms)

→ registration test

✓ should register SSV node Operator (487ms)

✓ should update ssv node Operator (198ms)

✓ add cluster (535ms)

Coverage:

The code coverage results were obtained by running npx hardhat coverage in the Nexus-Contracts project. We found the following results :

- Statements Coverage : 40.2%
- Branches Coverage : 43.16%
- Functions Coverage : 43.68%
- Lines Coverage : 48.26%

Conclusion:

The project offers a testing mechanism to improve the correctness of smart contracts; However, the test coverage percentage is low; it must be increased to cover all functionalities and test cases in order to guarantee the integrity of the code and the functionality of the protocol.

Re-Audit Coverage Results:

File	% Stmts	% Branch	% Funcs	% Lines
contracts	52.87	47.46	51.28	55
Nexus.sol	47.5	46.88	55	52.73
NodeOperator.sol	93.33	80	83.33	94.44
ValidatorExecutionRewards.sol	86.67	62.5	80	86.96
nexus_bridge	38.82	41.3	42.86	40.18
NexusBridgeDAO.sol	100	81.25	100	100
NexusBridgeUserCValue.sol	100	70	100	100
NexusBridgeUserRebase.sol	100	80	100	100
NexusDAIBridge.sol	0	0	0	0
nexusLibrary.sol	0	0	0	0
utils	46.15	33.33	58.33	52.17
NexusOwnable.sol	40	37.5	60	60
NexusProxy.sol	66.67	50	80	60
UUPSUpgreadable.sol	0	0	0	0

6 Conclusion

In this audit, we examined the design and implementation of Nexus Network contract and discovered several issues of varying severity. Nexus Network team addressed 12 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Nexus Network Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

7 Scope Files

7.1 Audit

Files	MD5 Hash
contracts/Nexus.sol	c0177752547d40350ede1da0a95af6b7
contracts/NodeOperator.sol	b0d4a1c03f6bac99892576dc44961bf7
contracts/ValidatorExecutionRewards.sol	f9c6dd01e8c761f8f58b232868ca9016
contracts/utils/NexusOwnable.sol	c2f5e243cd85d2f78f21552d9d129101
contracts/utils/NexusProxy.sol	5f9a9ea60f604ca4276ccf8c905aa628
contracts/utils/UUPSUpgreadable.sol	946dad0b4943a31d84f82f53ec86afe2
contracts/nexus_bridge/NexusBaseBridge.sol	0aa046cc75d34185e0ed97e3a0ff439b
contracts/nexus_bridge/NexusBridgeDAO.sol	1f9a13a304b07721bf4eacd052e2a71d
contracts/nexus_bridge/NexusBridgeUserCValue.sol	116d9e1136882cc580fc0d0e55298121
contracts/nexus_bridge/NexusBridgeUserRebase.sol	ac7c4e45dc9232754bdff3ce8aec2a99
contracts/nexus_bridge/NexusDAIBridge.sol	841e6f7b7b95296aae877716dce4ac18
contracts/nexus_bridge/nexusLibrary.sol	33e40f3130167e795ffe67a39ee7aa3a
create_keys.py	85b53a1e7ff1bc01ac93e56121c1e415
ssv/ssv_cli.py	a9982d68547e0335a2ec6f0522a8a361
ssv/ssv_dkg.py	288de73db52e9f62e860cd70164b12cb
ssv/ssv_utils.py	ffb6df8b2bd3a2d1751efc17c1a1552d

ssv/__init__.py	d41d8cd98f00b204e9800998ecf8427e
utils/aws.py	dad68b30506717655dcd82674cf4c89b
utils/contracts.py	e5a5f70f5cc3eb7da41a3fd1021768f4
utils/ethereum_connector.py	e776ec2f41c90b313e1ecb9715284d97
utils/helpers.py	285c8ce1aa5f1ed79739866a087eb4b7
utils/subgraph.py	922706535def7adb45a4c1ed7b1d8d7c
utils/types.py	37b7d94f60813ca67f518e67b49b10b3

7.2 Re-Audit

Files	MD5 Hash
contracts/Nexus.sol	c0177752547d40350ede1da0a95af6b7
contracts/NodeOperator.sol	b0d4a1c03f6bac99892576dc44961bf7
contracts/ValidatorExecutionRewards.sol	476eb0d925e0a3b52f723377706e234e
contracts/utils/NexusOwnable.sol	54a44787804b6277ed7e85d55031de13
contracts/utils/NexusProxy.sol	5f9a9ea60f604ca4276ccf8c905aa628
contracts/utils/UUPSUpgreadable.sol	946dad0b4943a31d84f82f53ec86afe2
contracts/nexus_bridge/NexusBaseBridge.sol	a2aba0cdec6c7f3990454f8aceeb28e1
contracts/nexus_bridge/NexusBridgeDAO.sol	3db7be2d1e9b602061f48c38574313ab
contracts/nexus_bridge/NexusBridgeUserCValue.sol	116d9e1136882cc580fc0d0e55298121

contracts/nexus_bridge/NexusBridgeUserRebase.sol	ac7c4e45dc9232754bdff3ce8aec2a99
contracts/nexus_bridge/NexusDAIBridge.sol	841e6f7b7b95296aae877716dce4ac18
contracts/nexus_bridge/nexusLibrary.sol	62b9c7b2562adf3bee03b2a8a3fbde4e
create_keys.py	848ab3a84f198adc0089161fcdf0de92
ssv/ssv_cli.py	a9982d68547e0335a2ec6f0522a8a361
ssv/ssv_dkg.py	288de73db52e9f62e860cd70164b12cb
ssv/ssv_utils.py	ffb6df8b2bd3a2d1751efc17c1a1552d
ssv/__init__.py	d41d8cd98f00b204e9800998ecf8427e
utils/aws.py	36d575d4b2917ecb8b83efbd082ceaf8
utils/contracts.py	e5a5f70f5cc3eb7da41a3fd1021768f4
utils/ethereum_connector.py	f18b4df81f2281fcc1cb9ee342e58b2b
utils/helpers.py	de449a290da495978854cd4566ad2719
utils/subgraph.py	3f2473b097660be1ce28de7b466afd90
utils/types.py	37b7d94f60813ca67f518e67b49b10b3

8 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com