BEOSIN
Blockchain Security

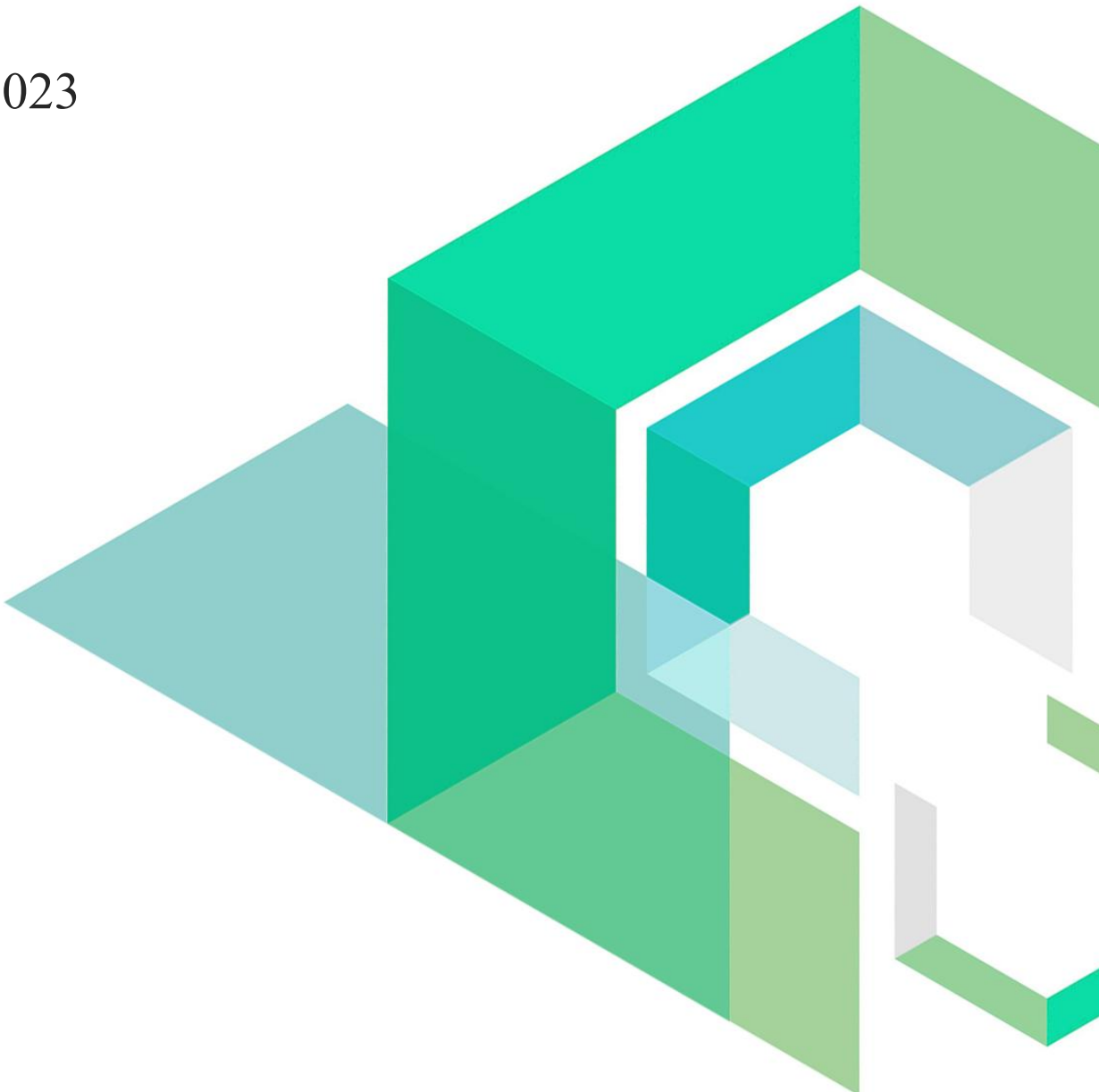
NodeDAO Protocol

Smart Contract Security Audit

V1.0

No. 202302011759

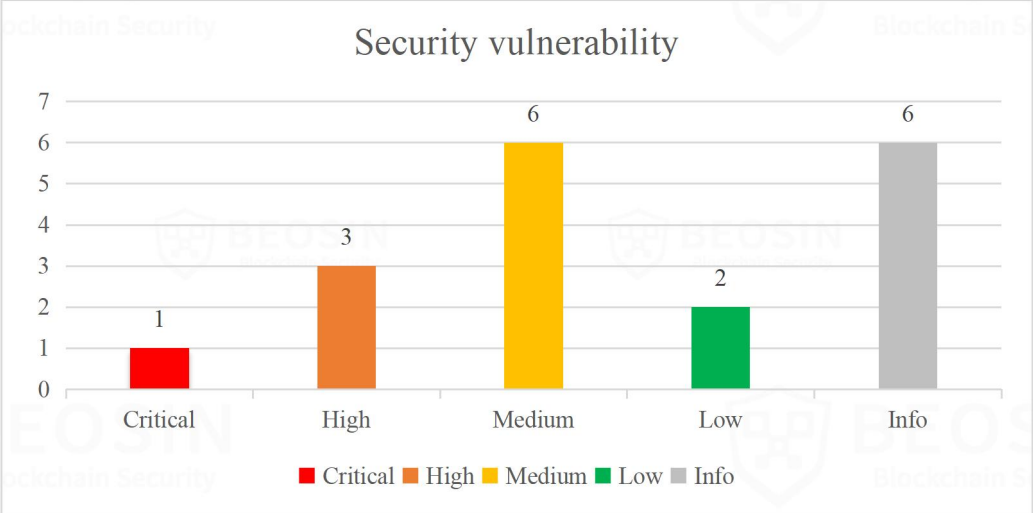
Feb 1st, 2023



Contents

Summary of Audit Results

After auditing, 1 Critical-risk, 3 High-risk, 6 Medium-risk, 2 Low-risk and 6 Info-risk items were identified in the NodeDAO Protocol project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Risk Description:**

1. Since the project is deployed in proxy mode, the audit is only for the implementation of the contract audit, the audit code details can be found in the following project fact sheet.

- **Project Description:**

1. **Business overview**

NodeDAO Protocol is a smart contract of liquid staking derivatives. This audit includes: Oracle module, Registry module, Rewards module, TimelockController module and staking module. There are two staking modes. The first staking mode is for users who stake less than 32 ETH. Users will get the corresponding NETH tokens and pay the corresponding fee when staking. The second staking model is for users who stake more than 32 ETH and they will receive the corresponding NFT tokens without paying any fee when staking. Users can enjoy the rewards by staking ETH to the contract and operator sends these amount to ETH 2.0. TimelockController module can permit controllers to delay the execution of transactions. The Rewards module is used to receive and settle user rewards. The Registry module provides a request to register an operator. Oracles module is used to obtain the latest data of the Beacon chain: the number of validators and ETH balance. The project administrator can add an address to Oracle Member, and Oracle Member can submit the latest data of the Beacon chain every day (the default is submitted once a day), when oracle member submits the same data more than 2/3, the latest data of beacon chain will be updated in this contract.

1 Overview

1.1 Project Overview

Project Name	NodeDAO Protocol
Platform	Ethereum
Audit scope	https://github.com/King-Hash-Org/NodeDAO-Protocol
Commit Hash	c97683960416e9ec1501e51a3b89fc97b3c27f40 6399e0c2c9c73af5ab9addc461a6ab90c006f788 cc2516ee23d9197ead6352d261484842f342fdc1 37182fa8f13480e1099d23bc7738d79539d804bf 281b22ae0f08039c597ea28a273e561382fddc36 1c62d5c0f4742b6af71ea358a9df6108cb90cd1f c08be80a7b2f42eae48e785cc856ca39c4a4b09c 04fd532096ccff06329ed3bf3619fb320ef4abbb 4abe7b261350f35c4895149c7ac2e64c99d6441e 030768f5ae2ecc89e7b5e1474c739d60220d47f3 d5e076bbd9264349fcdd56ddce80af24df61f1b9 356941569ff5e29763e5c639c5cf914a102fe437

1.2 Audit Overview

Audit work duration: Jan 16, 2023 – Feb 1, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
BeaconOracle-1	This function cannot update multiple cycles of beacon chain data	Critical	Fixed
BeaconOracle-2	The *addOracleMember* function is not designed properly	Low	Fixed
BeaconOracle-3	The data triggered by the event in the *reportBeacon* function is abnormal	Info	Fixed
BeaconOracle-4	An error occurred while compiling the contract	Info	Fixed
BeaconOracle-5	The *isOracleMember* function is not judged rigorously	Info	Fixed
ReportUtils-1	The *isReportDifferentAndCount* function is not designed properly	Medium	Fixed
NodeOperatorRegistry-1	This function can manipulate the controllerAddress id	Medium	Fixed
NodeOperatorRegistry-2	Related functions are missing the check for id.trusted	Info	Fixed
LiquidStaking-1	Contracts can't receive ETH transfers	High	Fixed
LiquidStaking-2	Inappropriate architecture design	High	Fixed
LiquidStaking-3	Existence arbitrage attack	High	Partially Fixed
LiquidStaking-4	The transfer address error in the *unwrapNFT* function	Medium	Fixed
LiquidStaking-5	Poorly designed *unstakeETH* function	Medium	Fixed
LiquidStaking-6	Array overflow	Low	Fixed
LiquidStaking-7	The setting of handling fees is unreasonable	Info	Fixed
LiquidStaking-8	Redundant code	Info	Fixed
ELVault-1	The *claimRewardsOfLiquidStaking* function is poorly designed	Medium	Fixed
ELVault-2	The *_settle* function is poorly designed	Medium	Fixed

Status Notes:

1. LiquidStaking-3 is Partially Fixed. The project party plans to use the robot to settle the reward, and then send the reward to the contract. When bot calls are infrequent, new users can enjoy the rewards from old users.

Finding Details:

[BeaconOracle-1] This function cannot update multiple cycles of beacon chain data

Severity Level	**Critical**
Type	Business Security
Lines	src/oracles/BeaconOracle.sol #L171-174
Description	When the first epoch report meets the quorum, isQuorum will change to true, but when the next expected epoch does not change the value to false, the beacon chain data for the next expected epoch will not be updated at this time.

```
165   function reportBeacon(
166       uint256 _epochId,
167       uint128 _beaconBalance,
168       uint32 _beaconValidators,
169       bytes32 _validatorRankingRoot
170   ) external {
171       if (isQuorum) {
172           emit achieveQuorum(_epochId, isQuorum, getQuorum());
173           return;
174       }
175       require(_epochId >= expectedEpochId, "EPOCH_IS_TOO_OLD");
176
177       // if expected epoch has advanced, check that this is the first epoch of the current frame
178       // and clear the last unsuccessful reporting
179       if (_epochId > expectedEpochId) {
180           require(_epochId == _getFrameFirstEpochOfDay(_getCurrentEpochId()), "UNEXPECTED_EPOCH");
181           _clearReportingAndAdvanceTo(_epochId);
182       }
183
```

Figure 1 Source code of *reportBeacon* function (Unfixed)

Recommendations	It is recommended to add related business logic functions for processing the next epoch cycle.
Status	Fixed. The project party has removed this part of the logic.

```
211   function reportBeacon(
212       uint256 _epochId,
213       uint128 _beaconBalance,
214       uint32 _beaconValidators,
215       bytes32 _validatorRankingRoot
216   ) external {
217       require(getCurrentEpochId() >= expectedEpochId, "EPOCH_IS_NOT_CURRENT_FRAME");
218       require(_epochId >= expectedEpochId, "EPOCH_IS_TOO_OLD");
219
220       // if expected epoch has advanced, check that this is the first epoch of the current frame
221       // and clear the last unsuccessful reporting
222       if (_epochId > expectedEpochId) {
223           require(_epochId == _getFrameFirstEpochOfDay(getCurrentEpochId()), "UNEXPECTED_EPOCH");
224           _clearReportingAndAdvanceTo(_epochId);
225       }
226
227       // make sure the oracle is from members list and has not yet voted
228       uint256 index = getMemberId(msg.sender);
229       require(index != MEMBER_NOT_FOUND, "MEMBER_NOT_FOUND");
230
231       uint256 bitMask = reportBitMaskPosition;
232       uint256 mask = 1 << index;
233       require(bitMask & mask == 0, "ALREADY_SUBMITTED");
234       // reported, set the bitmask to the specified bit
235       reportBitMaskPosition = bitMask | mask;
236
```

Figure 2 Source code of *reportBeacon* function (Fixed)

[BeaconOracle-2] The *addOracleMember* function is not designed properly

Severity Level	Low
Type	Business Security
Lines	src/oracles/BeaconOracle.sol#100-107
Description	BeaconOracle contract *addOracleMember* function, add oracle Member, always add at the end of the array, so there may be oracleMemberCount is not greater than the maximum number of MEMBERS (part of the oracle member was removed), but the last index of oracleMembers has exceeded 255, which will cause an exception to the voting for the MEMBERS who index exceeds 255.

```
100     function addOracleMember(address _oracleMember) external onlyDao {
101         require(address(0) != _oracleMember, "BAD_ARGUMENT");
102         require(oracleMemberCount < MAX_MEMBERS, "TOO_MANY_MEMBERS");
103         require(MEMBER_NOT_FOUND == _getMemberId(_oracleMember), "MEMBER_EXISTS");
104
105         oracleMembers.push(_oracleMember);
106         oracleMemberCount++;
107
108         emit AddOracleMember(_oracleMember);
109     }
```

Figure 3 Source code of *addOracleMember* function (Unfixed)

Recommendations	It is recommended to fill the corresponding index data that has been deleted when adding members.
Status	Fixed.

```
132     */
133     function addOracleMember(address _oracleMember) external onlyDao {
134         require(address(0) != _oracleMember, "BAD_ARGUMENT");
135         require(MEMBER_NOT_FOUND == getMemberId(_oracleMember), "MEMBER_EXISTS");
136
137         bool isAdd = false;
138         for (uint256 i = 0; i < oracleMembers.length; ++i) {
139             if (oracleMembers[i] == address(0)) {
140                 oracleMembers[i] = _oracleMember;
141                 isAdd = true;
142                 break;
143             }
144         }
145
146         if (!isAdd) {
147             oracleMembers.push(_oracleMember);
148         }
149
150         oracleMemberCount++;
151
152         emit AddOracleMember(_oracleMember);
153     }
```

Figure 4 Source code of *addOracleMember* function (Fixed)

[BeaconOracle-3] The data triggered by the event in the *reportBeacon* function is abnormal

Severity Level	Info
Type	Business Security
Lines	src/oracles/BeaconOracle.sol #L220 src/oracles/BeaconOracle.sol #L230
Description	The number of votes for calling the *reportBeacon* function is not added when triggering the report success event.

```
217         if (i < currentReportVariants.length) {
218             if (sameCount + 1 >= quorum) {
219                 _dealReport(nextEpochId, _beaconBalance, _beaconValidators, _validatorRankingRoot);
220                 emit ReportSuccess(_epochId, quorum, sameCount);
221             } else {
222                 // increment report counter, see ReportUtils for details
223                 currentReportVariants[i] = ReportUtils.compressReportData(
224                     _validatorRankingRoot, _beaconBalance, _beaconValidators, sameCount + 1
225                 );
226             }
227         } else {
228             if (quorum == 1) {
229                 _dealReport(nextEpochId, _beaconBalance, _beaconValidators, _validatorRankingRoot);
230                 emit ReportSuccess(_epochId, quorum, sameCount);
231             } else {
232                 currentReportVariants.push(
233                     ReportUtils.compressReportData(
234                         _validatorRankingRoot, _beaconBalance, _beaconValidators, sameCount + 1
235                     )
236                 );
237             }
```

Figure 5 Source code of *reportBeacon* function (Unfixed)

Recommendations	It is recommended to change sameCount to sameCount+1.
Status	Fixed.

```
237
238         if (i < currentReportVariants.length) {
239             if (sameCount + 1 >= quorum) {
240                 _dealReport(nextEpochId, _beaconBalance, _beaconValidators, _validatorRankingRoot);
241                 emit ReportSuccess(_epochId, quorum, sameCount + 1);
242             } else {
243                 // increment report counter, see ReportUtils for details
244                 currentReportVariants[i] = ReportUtils.compressReportData(
245                     _validatorRankingRoot, _beaconBalance, _beaconValidators, sameCount + 1
246                 );
247             }
248         } else {
249             if (quorum == 1) {
250                 _dealReport(nextEpochId, _beaconBalance, _beaconValidators, _validatorRankingRoot);
251                 emit ReportSuccess(_epochId, quorum, sameCount + 1);
252             } else {
253                 currentReportVariants.push(
254                     ReportUtils.compressReportData(
255                         _validatorRankingRoot, _beaconBalance, _beaconValidators, sameCount + 1
256                     )
257                 );
258             }
259         }
260     }
```

Figure 6 Source code of *reportBeacon* function (Fixed)

[BeaconOracle-4] An error occurred while compiling the contract

Severity Level	Info
Type	Coding Conventions
Lines	src/oracles/BeaconOracle.sol
Description	An error occurs when compiling the contract with the 0.8.7 version of the compiler, which will cause the contract to fail to compile.
Recommendations	It is recommended to fix the compiler version to version 0.8.8.
Status	Fixed.

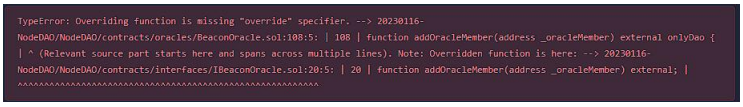


Figure 7 Compile Error

[BeaconOracle-5] The isOracleMember function is not judged rigorously

Severity Level	Info
Type	Business Security
Lines	src/oracles/BeaconOracle.sol#131-132
Description	The *isOracleMember* function may return true if the address passed in is 0.

```
130     */
131     function isOracleMember(address _oracleMember) external view returns (bool) {
132         return _isOracleMember(_oracleMember);
133     }
134
```

Figure 8 Source code of *isOracleMember* function (Unfixed)

Recommendations	It is recommended to add a review (_ORACLEMEMBER! = Address (0), "Address Provided Invalid");
Status	Fixed.

```
175     */
176     function isOracleMember(address _oracleMember) external view returns (bool) {
177         require(address(0) != _oracleMember, "BAD_ARGUMENT");
178         return _isOracleMember(_oracleMember);
179     }
```

Figure 9 Source code of *isOracleMember* function (Fixed)

[ReportUtils-1] The *isReportDifferentAndCount* function is not designed properly

Severity Level	Medium
Type	Business Security
Lines	src\oracles\ReportUtils.sol #L38
Description	The judgment logic shows that it is only true when all the parameters of _validatorRankingRoot, _beaconBalance and _beaconValidators are different, which will cause an exception when updating the same number of votes.

```
function isReportDifferentAndCount(
    bytes memory value,
    bytes32 _validatorRankingRoot,
    uint128 _beaconBalance,
    uint32 _beaconValidators
) internal pure returns (bool, uint16) {
    (bytes32 root, uint128 balance, uint32 validators, uint16 sameCount) = decompressReportData(value);
    bool isDifferent = root != _validatorRankingRoot && balance != _beaconBalance && validators != _beaconVal
    return (isDifferent, sameCount);
}
```

Figure 10 Source code of *isReportDifferentAndCount* function (Unfixed)

```
while (i < currentReportVariants.length) {
    (bool isDifferent, uint16 count) = ReportUtils.isReportDifferentAndCount(
        currentReportVariants[i], _validatorRankingRoot, _beaconBalance, _beaconValidators
    );

    if (isDifferent) {
        ++i;
    } else {
        sameCount = count;
        break;
    }
}
```

Figure 11 Source code of *reportBeacon* function

Recommendations	It is suggested to change the logic to false as long as one of the parameters is different, i.e. bool isDifferent = !(root == _validatorRankingRoot && balance == _beaconBalance && validators == _beaconValidators);
Status	Fixed.

```
30      */
31      function isReportDifferentAndCount(
32          bytes memory value,
33          bytes32 _validatorRankingRoot,
34          uint128 _beaconBalance,
35          uint32 _beaconValidators
36      ) internal pure returns (bool, uint16) {
37          (bytes32 root, uint128 balance, uint32 validators, uint16 sameCount) = decompressReportData(value);
38          bool isDifferent =
39              !(root == _validatorRankingRoot && balance == _beaconBalance && validators == _beaconValidators);
40          return (isDifferent, sameCount);
41      }
42  }
43
```

Figure 12 Source code of *isReportDifferentAndCount* function (Fixed)

[NodeOperatorRegistry-1] This function can manipulate the controllerAddress id

Severity Level	Medium
Type	Business Security
Lines	src\registries\NodeOperatorRegistry.sol #L172-178
Description	Since the controllerAddress of multiple operators can be modified to the same address, malicious attackers will modify the controllerAddress to the controllerAddress of other operators, and then modify the trustedControllerAddress[_controllerAddress] to a malicious id.

```
172   function setNodeOperatorControllerAddress(uint256 _id, address _controllerAddress) external operatorExists(_id) {
173       NodeOperator memory operator = operators[_id];
174       require(msg.sender == operator.controllerAddress || msg.sender == dao, "AUTH_FAILED");
175       trustedControllerAddress[operator.controllerAddress] = 0;
176       operators[_id].controllerAddress = _controllerAddress;
177       trustedControllerAddress[_controllerAddress] = _id;
178       emit NodeOperatorControllerAddressSet(_id, operator.name, _controllerAddress);
179   }
```

Figure 13 Source code of *setNodeOperatorControllerAddress* function (Unfixed)

Recommendations	It is recommended to add a mapping variable storage and determine whether the controllerAddress is used. That is, one controllerAddress can only control one operator.
Status	Fixed.

```
180   function setNodeOperatorControllerAddress(uint256 _id, address _controllerAddress) external operatorExists(_id) {
181       require(!usedControllerAddress[_controllerAddress], "controllerAddress is used");
182
183       NodeOperator memory operator = operators[_id];
184       require(msg.sender == operator.controllerAddress || msg.sender == dao, "AUTH_FAILED");
185       if (trustedControllerAddress[operator.controllerAddress] == _id) {
186           trustedControllerAddress[operator.controllerAddress] = 0;
187           trustedControllerAddress[_controllerAddress] = _id;
188       }
189
190       operators[_id].controllerAddress = _controllerAddress;
191       usedControllerAddress[_controllerAddress] = true;
192
193       emit NodeOperatorControllerAddressSet(_id, operator.name, _controllerAddress);
194   }
```

Figure 14 Source code of *setNodeOperatorControllerAddress* function (Fixed)

[NodeOperatorRegistry-2] Related functions are missing the check for id.trusted

Severity Level	Info
Type	Business Security
Lines	src\registries\NodeOperatorRegistry.sol #L121-138
Description	These functions don't constrain to repeate operation from same _id. Repeatedly adds and removes the same _id will also update TotalTrustedOperators. If the same _id is removed repeatedly, when TotalTrustedOperators are 0, other _id will not be removed again.

```
121     function setTrustedOperator(uint256 _id) external onlyDao operatorExists(_id) {
122         NodeOperator memory operator = operators[_id];
123         operators[_id].trusted = true;
124         totalTrustedOperators += 1;
125         trustedControllerAddress[operator.controllerAddress] = _id;
126         emit NodeOperatorTrustedSet(_id, operator.name, true);
127     }
128
129     /**
130      * @notice Remove an operator as trusted
131      * @param _id operator id
132      */
133     function removeTrustedOperator(uint256 _id) external onlyDao operatorExists(_id) {
134         NodeOperator memory operator = operators[_id];
135         operators[_id].trusted = false;
136         totalTrustedOperators -= 1;
137         trustedControllerAddress[operator.controllerAddress] = 0;
138         emit NodeOperatorTrustedRemove(_id, operator.name, false);
139     }
```

Figure 15 Related function code screenshot (Unfixed)

Recommendations	It is recommended to judge when adding and removing to avoid updating the value of TotalTrustedOperators when adding and removing repeatedly.
Status	Fixed.

```
127     function setTrustedOperator(uint256 _id) external onlyDao operatorExists(_id) {
128         NodeOperator memory operator = operators[_id];
129         require(!operator.trusted, "The operator is already trusted");
130         operators[_id].trusted = true;
131         totalTrustedOperators += 1;
132         trustedControllerAddress[operator.controllerAddress] = _id;
133         emit NodeOperatorTrustedSet(_id, operator.name, true);
134     }
135
136     /**
137      * @notice Remove an operator as trusted
138      * @param _id operator id
139      */
140     function removeTrustedOperator(uint256 _id) external onlyDao operatorExists(_id) {
141         NodeOperator memory operator = operators[_id];
142         require(operator.trusted, "operator is not trusted");
143         operators[_id].trusted = false;
144         totalTrustedOperators -= 1;
145         trustedControllerAddress[operator.controllerAddress] = 0;
146         emit NodeOperatorTrustedRemove(_id, operator.name, false);
147     }
```

Figure 16 Related function code screenshot (Fixed)

[LiquidStaking-1] Contracts can't receive ETH transfers

Severity Level	**High**
Type	Business Security
Lines	src\LiquidStaking.sol
Description	The absence of a function to receive ETH transfers in the LiquidStaking contract will cause the vault contract to fail to send rewards.
Recommendations	It is recommended to add the *receive* function to receive ETH transfer.
Status	Fixed.

```
535
536     receive() external payable {}
537 }
538
```

Figure 17 Add screenshot of *receive* function

[LiquidStaking-2] Inappropriate architecture design

Severity Level	**High**
Type	Business Security
Lines	src\LiquidStaking.sol
Description	The *registerValidator* function needs to consume the balance of the corresponding operatorId, that is, the value of the variable operatorPoolBalances[operatorId]. In the *stakeETH* and *unstakeETH* functions, when the user calls the *stakeETH* function to stake, operatorPoolBalances[_operatorId] will increase according to the amount of ETH sent, and send NETH certificate tokens to the user; when calling *unstakeETH* to extract ETH, it will be based on the NETH certificate To withdraw the ETH in the unstakePoolBalances pool, but the operatorPoolBalances[operatorId] has not been updated, user can use this part of the extra operatorPoolBalances to register validator.
Recommendations	It is recommended to record the stake amount of the user on the corresponding operator. When the user withdraws ETH, reduce the stake amount of the user on the corresponding operator, and update the pool balance of the operator synchronously.
Status	Fixed. This code logic has been removed by the project party.

[LiquidStaking-3] Existence arbitrage attack

Severity Level	**High**
Type	Business Security
Lines	src\LiquidStaking.sol
Description	Since the *stakeETH* function calculates the amount of NETH based on the beaconBalance and address(this).balance, but the rewarded ETH is put into the ELVault contract, which will cause arbitrage attack. For example, the current ETH and NETH exchange rate is 1:1. The pool reserve is 100 ETH and 100 NETH. Here the attacker can buy 100 NETH first, then the total pool is 200 ETH and 200 NETH. then use the batchClaimRewardsOfOperator function to take the rewarded 20 ETH into the contract. At this point, the pool reserve is 220 ETH and 200 NETH. The attacker can take 100 NETH and exchange it for 110 ETH.
Recommendations	It is recommended to complete all the rewards and send it to the LiquidStaking contract, and then calculate the value of ETH/NETH.
Status	Partially Fixed. Project party reply: Use bots to help users claim rewards offline. partially fixed here is because the project party has removed the withdrawal function of withdrawal, so it is not easy to use arbitrage attacks. When the robot calls are not frequent, new users can enjoy the rewards from old users.

[LiquidStaking-4] The transfer address error in the *unwrapNFT* function

Severity Level	Medium
Type	Business Security
Lines	src\LiquidStaking.sol #326
Description	The wrong transfer address in the *unwrapNFT* function should be that the user sends NFT tokens to the contract and the contract sends NETH to the user.

```
305  function unwrapNFT(uint256 tokenId, bytes32[] memory proof, uint256 value) external nonReentrant {
306      uint256 operatorId = vNFTContract.operatorOf(tokenId);
307
308      bool trusted;
309      address vaultContractAddress;
310      (trusted,,,, vaultContractAddress) = nodeOperatorRegistryContract.getNodeOperator(operatorId, false);
311      require(trusted, "permission denied");
312
313      bytes memory pubkey = vNFTContract.validatorOf(tokenId);
314      bool success = beaconOracleContract.verifyNftValue(proof, pubkey, value, tokenId);
315      require(success, "verifyNftValue fail");
316
317      uint256 amountOut = _getNethOut(value, 0);
318
319      _liquidUserNfts[tokenId] = false;
320
321      _settle(operatorId);
322      claimRewardsOfUser(tokenId);
323
324      vNFTContract.safeTransferFrom(msg.sender, address(this), tokenId);
325
326      success = nETHContract.transferFrom(msg.sender, address(this), amountOut);
327      require(success, "Failed to transfer neth");
328
329      IELVault(vaultContractAddress).setUserNft(tokenId, 0);
330      nftWrapNonce = nftWrapNonce + 1;
331
332      emit NftUnwrap(tokenId, operatorId, value, amountOut);
333  }
```

Figure 18 Source code of *unwrapNFT* function (Unfixed)

Recommendations	It is recommended to replace the sequence of msg.sender and address(this).
Status	Fixed.

```
290  //5. Set the vault contract setUserNft to 0
291  function unwrapNFT(uint256 tokenId, bytes32[] memory proof, uint256 value) external nonReentrant {
292      uint256 operatorId = vNFTContract.operatorOf(tokenId);
293
294      bool trusted;
295      address vaultContractAddress;
296      (trusted,,,, vaultContractAddress) = nodeOperatorRegistryContract.getNodeOperator(operatorId, false);
297      require(trusted, "permission denied");
298
299      bytes memory pubkey = vNFTContract.validatorOf(tokenId);
300      bool success = beaconOracleContract.verifyNftValue(proof, pubkey, value, tokenId);
301      require(success, "verifyNftValue fail");
302
303      uint256 amountOut = _getNethOut(value, 0);
304
305      _liquidUserNfts[tokenId] = false;
306
307      claimRewardsOfUser(tokenId);
308      vNFTContract.safeTransferFrom(msg.sender, address(this), tokenId);
309      // success = nETHContract.transferFrom(address(this), msg.sender, amountOut);
310      success = nETHContract.transfer(msg.sender, amountOut);
311      require(success, "Failed to transfer neth");
312
313      IELVault(vaultContractAddress).setUserNft(tokenId, 0);
314      nftWrapNonce = nftWrapNonce + 1;
315
316      emit NftUnwrap(tokenId, operatorId, value, amountOut);
317  }
```

Figure 19 Source code of *unwrapNFT* function (Fixed)

[LiquidStaking-5] Poorly designed *unstakeETH* function

Severity Level	Medium
Type	Business Security
Lines	src\LiquidStaking.sol#168
Description	The quantity transferred in the *unstakeETH* function is amountOut, which will result in exiting the collateral with no fee.

```
151     function unstakeETH(uint256 amount) external nonReentrant {
152         uint256 amountOut = getEthOut(amount);
153         require(unstakePoolBalances >= amountOut, "UNSTAKE_POOL_INSUFFICIENT_BALANCE");
154
155         nETHContract.whiteListBurn(amount, msg.sender);
156         unstakePoolBalances -= amountOut;
157
158         uint256 userAmount;
159         uint256 feeAmount;
160         if (unstakeFeeRate == 0) {
161             userAmount = amountOut;
162         } else {
163             feeAmount = amountOut * unstakeFeeRate / totalBasisPoints;
164             userAmount = amountOut - feeAmount;
165             transfer(feeAmount, daoVaultAddress);
166         }
167
168         transfer(amountOut, msg.sender);
169
170         emit EthUnstake(msg.sender, amount, amountOut);
171     }
```

Figure 20 Source code of *unstakeETH* function (Unfixed)

Recommendations	Suggest replacing amountOut with userAmount.
Status	Fixed.

```
130
131     function unstakeETH(uint256 amount) external nonReentrant {
132         require(false, "Not supported yet");
133     }
134
```

Figure 21 Source code of *unstakeETH* function (Fixed)

[LiquidStaking-6] Array overflow

Severity Level	Low
Type	General Vulnerability
Lines	src\LiquidStaking.sol# 452-495
Description	Because in *getliquidnfts* and *getOperatorNFTS* functions, liquidnfts are defined as an array of specified length, and the array of this length is based on the length of the array of _liquidnfts. This will cause the value of i to exceed the array length of liquidnfts. *GetOperatorNFTS* function is the same problem.

```
448     // Validators currently owned by the stake pool
449     function getLiquidNfts() public view returns (uint256[] memory) {
450         uint256 nftCount;
451         uint256[] memory liquidNfts;
452         uint256 i;
453         for (i = 0; i < _liquidNfts.length; i++) {
454             uint256 tokenId = _liquidNfts[i];
455             if (_liquidUserNfts[tokenId]) {
456                 nftCount += 1;
457             }
458         }
459
460         liquidNfts = new uint256[] (nftCount);
461         for (i = 0; i < _liquidNfts.length; i++) {
462             uint256 tokenId = _liquidNfts[i];
463             if (_liquidUserNfts[tokenId]) {
464                 liquidNfts[i] = tokenId;
465             }
466         }
467
468         return liquidNfts;
469     }
470
471     function getOperatorNfts(uint256 operatorId) public view returns (uint256[] memory) {
472         uint256 nftCount;
473         uint256[] memory operatorNfts;
474
475         uint256[] memory nfts = _operatorNfts[operatorId];
476         uint256 i;
477         for (i = 0; i < nfts.length; i++) {
478             uint256 tokenId = nfts[i];
479             if (_liquidUserNfts[tokenId]) {
480                 nftCount += 1;
481             }
482         }
483
484         operatorNfts = new uint256[] (nftCount);
485         for (i = 0; i < nfts.length; i++) {
486             uint256 tokenId = nfts[i];
487             if (_liquidUserNfts[tokenId]) {
488                 operatorNfts[i] = tokenId;
489             }
490         }
491
492         return operatorNfts;
493     }
494
```

Figure 22 Related code screenshot (Unfixed)

Recommendations	It is recommended to define a variable to traverse the receiving array data.
Status	Fixed.

```
433     function getLiquidNfts() public view returns (uint256[] memory) {
434         uint256 nftCount;
435         uint256[] memory liquidNfts;
436         uint256 i;
437         for (i = 0; i < _liquidNfts.length; i++) {
438             uint256 tokenId = _liquidNfts[i];
439             if (_liquidUserNfts[tokenId]) {
440                 nftCount += 1;
441             }
442         }
443
444         liquidNfts = new uint256[] (nftCount);
445         uint256 j;
446         for (i = 0; i < _liquidNfts.length; i++) {
447             uint256 tokenId = _liquidNfts[i];
448             if (_liquidUserNfts[tokenId]) {
449                 liquidNfts[j] = tokenId;
450                 j += 1;
451             }
452         }
453
454         return liquidNfts;
455     }
456
457     function getOperatorNfts(uint256 operatorId) public view returns (uint256[] memory) {
458         uint256 nftCount;
459         uint256[] memory operatorNfts;
460
461         uint256[] memory nfts = _operatorNfts[operatorId];
462         uint256 i;
463         for (i = 0; i < nfts.length; i++) {
464             uint256 tokenId = nfts[i];
465             if (_liquidUserNfts[tokenId]) {
466                 nftCount += 1;
467             }
468         }
469
470         operatorNfts = new uint256[] (nftCount);
471         uint256 j;
472         for (i = 0; i < nfts.length; i++) {
473             uint256 tokenId = nfts[i];
474             if (_liquidUserNfts[tokenId]) {
475                 operatorNfts[j] = tokenId;
476                 j += 1;
477             }
478         }
479
480         return operatorNfts;
481     }
```

Figure 23 Related code screenshot (Fixed)

[LiquidStaking-7] The setting of handling fees is unreasonable

Severity Level	Info
Type	Business Security
Lines	src\LiquidStaking.sol# 506-514
Description	Since the rate range is limited to less than one hundred percent in the *setUnstakePoolSize* and *setUnstakeFeeRate* functions, it is still possible to set the rate to a higher value, which may result in higher than expected fees being charged.

```
506     function setDepositFeeRate(uint256 _feeRate) external onlyDao {
507         require(_feeRate < totalBasisPoints, "cannot be 100%");
508         depositFeeRate = _feeRate;
509     }
510
511     function setUnstakeFeeRate(uint256 _feeRate) external onlyDao {
512         require(_feeRate < totalBasisPoints, "cannot be 100%");
513         unstakeFeeRate = _feeRate;
514     }
```

Figure 24 Related code screenshot (Unfixed)

Recommendations	It is recommended to set a reasonable range.
Status	Fixed.

```
494
495     function setDepositFeeRate(uint256 _feeRate) external onlyDao {
496         require(_feeRate <= 1000, "Rate too high");
497         depositFeeRate = _feeRate;
498     }
499
500     function setUnstakeFeeRate(uint256 _feeRate) external onlyDao {
501         require(_feeRate <= 1000, "Rate too high");
502         unstakeFeeRate = _feeRate;
503     }
```

Figure 25 Related code screenshot (Fixed)

[LiquidStaking-8] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	src\LiquidStaking.sol# 242-250
Description	The following functions are redundant code. It is recommended to delete it.

```
241    //slither-disable-next-line reentrancy-events
242    function deposit(bytes calldata data) private {
243        bytes calldata pubkey = data[16:64];
244        bytes calldata withdrawalCredentials = data[64:96];
245        bytes calldata signature = data[96:192];
246        bytes32 depositDataRoot = bytes32(data[192:224]);
247
248        depositContract.deposit{value: 32 ether}(pubkey, withdrawalCredentials, signature, depositDataRoot);
249
250        emit Eth32Deposit(pubkey, withdrawalCredentials, msg.sender);
251    }
```

Figure 26 Source code of *deposit* function (Unfixed)

Recommendations	It is recommended to delete.
Status	Fixed. The project has removed this code section.

[ELVault-1] The *claimRewardsOfLiquidStaking* function is poorly designed

Severity Level	Medium
Type	Business Security
Lines	src\rewards\ELVault.sol# 201-208
Description	In the *claimRewardsOfLiquidStaking* function, the update of the unclaimedRewards parameter is missing. The unclaimedRewards records unclaimed rewards, and failure to update it will lead to confusion in the reward calculation.

```
201    function claimRewardsOfLiquidStaking() external nonReentrant onlyLiquidStaking returns (uint256) {
202        uint256 nftRewards = liquidStakingReward;
203        liquidStakingReward = 0;
204        transfer(nftRewards, liquidStakingContract);
205
206        emit RewardClaimed(liquidStakingContract, nftRewards);
207
208        return nftRewards;
209    }
210
```

Figure 27 Source code of *claimRewardsOfLiquidStaking* function (Unfixed)

Recommendations	Suggest updating the unclaimedRewards parameter.
Status	Fixed.

```
203    function claimRewardsOfLiquidStaking() external nonReentrant onlyLiquidStaking returns (uint256) {
204        uint256 nftRewards = liquidStakingReward;
205        unclaimedRewards -= nftRewards;
206        liquidStakingReward = 0;
207        transfer(nftRewards, liquidStakingContract);
208
209        emit RewardClaimed(liquidStakingContract, nftRewards);
210
211        return nftRewards;
212    }
213
```

Figure 28 Source code of *claimRewardsOfLiquidStaking* function (Fixed)

[ELVault-2] The *_settle* function is poorly designed

Severity Level	Medium
Type	Business Security
Lines	src\rewards\ELVault.sol# 109
Description	In the *_settle* function, the new reward is added by "address(this).balance - unclaimedRewards - operatorRewards;" however, the contract also contains the daoRewards part, and in the next settlement of the reward address(this).balance will include the daoRewards part of the reward, resulting in confusion in the reward calculation.

```
107     */
108     function _settle() private {
109         uint256 outstandingRewards = address(this).balance - unclaimedRewards - operatorRewards;
110         if (outstandingRewards == 0 || cumArr[cumArr.length - 1].height == block.number) {
111             return;
112         }
113
114         uint256 comission = (outstandingRewards * comissionRate) / 10000;
115         uint256 daoReward = (comission * daoComissionRate) / 10000;
116         daoRewards += daoReward;
117         operatorRewards += comission - daoReward;
118
119         outstandingRewards -= comission;
120         unclaimedRewards += outstandingRewards;
121
122         uint256 operatorNftCounts = vNFTContract.getNftCountsOfOperator(operatorId);
123         uint256 averageRewards = outstandingRewards / operatorNftCounts;
124
125         liquidStakingReward += averageRewards * (operatorNftCounts - userNftsCount);
126
127         uint256 currentValue = cumArr[cumArr.length - 1].value + averageRewards;
128         RewardMetadata memory r = RewardMetadata({value: currentValue, height: block.number});
129         cumArr.push(r);
130
131         emit Settle(block.number, averageRewards);
132     }
133
```

Figure 29 Source code of *_settle* function (Unfixed)

Recommendations	Suggest address(this).balance minus daoRewards.
Status	Fixed.

```
107     */
108     function _settle() internal {
109         uint256 outstandingRewards = address(this).balance - unclaimedRewards - operatorRewards - daoRewards;
110         if (outstandingRewards == 0 || cumArr[cumArr.length - 1].height == block.number) {
111             return;
112         }
113
114         uint256 comission = (outstandingRewards * comissionRate) / 10000;
115         uint256 daoReward = (comission * daoComissionRate) / 10000;
116         daoRewards += daoReward;
117         operatorRewards += comission - daoReward;
118
119         outstandingRewards -= comission;
120         unclaimedRewards += outstandingRewards;
121
122         uint256 operatorNftCounts = vNFTContract.getNftCountsOfOperator(operatorId);
123         uint256 averageRewards = outstandingRewards / operatorNftCounts;
124
125         liquidStakingReward += averageRewards * (operatorNftCounts - userNftsCount);
126
127         uint256 currentValue = cumArr[cumArr.length - 1].value + averageRewards;
128         RewardMetadata memory r = RewardMetadata({value: currentValue, height: block.number});
129         cumArr.push(r);
130
131         emit Settle(block.number, averageRewards);
132     }
133
```

Figure 30 Source code of *_settle* function (Fixed)

Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com