Competitive Security Assessment

StakestoneEigenlayerHelper

Mar 18th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3′s Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

• Verify the code base is compliant with the most up-to-date industry standards and security best practices.

• Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

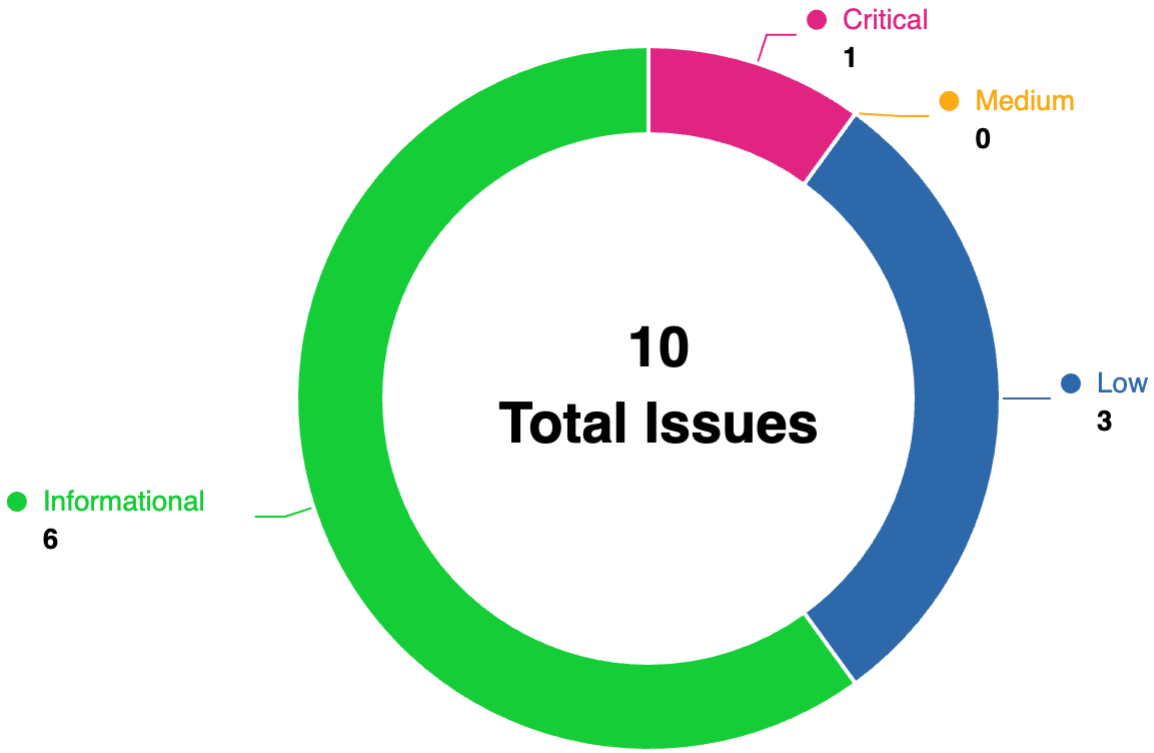
Overview

Project Name	StakestoneEigenlayerHelper
Language	Solidity
Codebase	• https://github.com/stakestone/stone-vault-v1 • audit version - ccab0ca8db37c759a289402c8fbb109266c375b7 • final version - 9e634b3a5de001f5102062362833c3193a0e1b83
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/interfaces/IBatchDeposit.sol	0c2c9518ee251c62c4dde8884db8bce836ce9f8386e9cc3652657bfee98a28ef
contracts/interfaces/IEigenPod.sol	f07cbbae5e03eef12f8998735655450734 5731ec4b644ccf2163a9e1d84bd48a
contracts/interfaces/IEigenPodManager.sol	8c1939f7fcf141741e50d65f8cae65295814bb75a4192006fabed7ffdf1a1392
contracts/strategies/eigen/Account.sol	33229f15ca879e5504c7b5bd95ae2b40871db9bc96b190482742e39b346dff3f
contracts/strategies/eigen/EigenNativeRestakingStrategy.sol	3eaf851d2b4ba94ea26db191fabd6b8478d04439736bdf9d0d629ade7528f87e

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
STO-1	Incorrect call of the function `claimDelayedWithdrawals()`	Logical	Critical	Fixed	thereksfour, Yaodao, biakia
STO-2	Potential array out-of-bounds error	Logical	Low	Fixed	8olidity, biakia
STO-3	Ownership change should use two-step process	Privilege Related	Low	Fixed	ravikiran_web3, Yaodao, biakia
STO-4	EigenNativeRestakingStrategy::instantWithdraw() should not have notAtSameBlock modifier to allow instant withdrawal	Logical	Low	Fixed	ravikiran_web3
STO-5	Variables that could be declared as immutable	Language Specific	Informational	Fixed	biakia
STO-6	Unused variables and imports	Code Style	Informational	Fixed	ravikiran_web3, biakia
STO-7	Unused function param	Code Style	Informational	Fixed	ravikiran_web3, 0xac, biakia, Yaodao

STO-8	Replace abi.encodeWithSelector by abi.encodeCall in several functions in contract EigenNativeRestakingStrategy	Language Specific	Informational	Fixed	ginlee
STO-9	Lack of check `address(0)`	Logical	Informational	Fixed	8olidity, biakia
STO-10	Be cautious when using block.timestamp in Arbitrum	Logical	Informational	Acknowledged	ginlee

STO-1:Incorrect call of the function `claimDelayedWithdrawals()`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	thereksfour, Yaodao, bi akia

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L171-L190
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L171-190
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L171-L190

```
171: function finalizeWithdrawingNode(
172:         uint256 _nodeAmount,
173:         address _eigenPod
174:     ) external onlyGovernance {
175:         address podOwner = eigenPodOwners[_eigenPod];
176:         require(podOwner != address(0), "EigenPod not exist");
177:
178:         Account account = Account(payable(podOwner));
179:         account.invoke(
180:             _eigenPod,
181:             0,
182:             abi.encodeWithSelector(
183:                 IEigenPod.claimDelayedWithdrawals.selector,
184:                 type(uint256).max
185:             )
186:         );
187:         account.invoke(address(this), podOwner.balance, "");
188:
189:         withdrawingNodeAmount -= _nodeAmount;
190:     }
```


```
171: function finalizeWithdrawingNode(
172:         uint256 _nodeAmount,
173:         address _eigenPod
174:     ) external onlyGovernance {
175:         address podOwner = eigenPodOwners[_eigenPod];
176:         require(podOwner != address(0), "EigenPod not exist");
177:
178:         Account account = Account(payable(podOwner));
179:         account.invoke(
180:             _eigenPod,
181:             0,
182:             abi.encodeWithSelector(
183:                 IEigenPod.claimDelayedWithdrawals.selector,
184:                 type(uint256).max
185:             )
186:         );
187:         account.invoke(address(this), podOwner.balance, "");
188:
189:         withdrawingNodeAmount -= _nodeAmount;
190:     }
```

```
171: function finalizeWithdrawingNode(
172:         uint256 _nodeAmount,
173:         address _eigenPod
174:     ) external onlyGovernance {
175:         address podOwner = eigenPodOwners[_eigenPod];
176:         require(podOwner != address(0), "EigenPod not exist");
177:
178:         Account account = Account(payable(podOwner));
179:         account.invoke(
180:             _eigenPod,
181:             0,
182:             abi.encodeWithSelector(
183:                 IEigenPod.claimDelayedWithdrawals.selector,
184:                 type(uint256).max
185:             )
186:         );
187:         account.invoke(address(this), podOwner.balance, "");
188:
189:         withdrawingNodeAmount -= _nodeAmount;
190:     }
```

Description

thereksfour: EigenNativeRestakingStrategy.finalizeWithdrawingNode() will call _eigenPod.claimDelayedWithdrawals() to finalize the ETH withdrawal. However there is no claimDelayedWithdrawals() method in _eigenPod.

```
    function finalizeWithdrawingNode(
        uint256 _nodeAmount,
        address _eigenPod
    ) external onlyGovernance {
        address podOwner = eigenPodOwners[_eigenPod];
        require(podOwner != address(0), "EigenPod not exist");

        Account account = Account(payable(podOwner));
        account.invoke(
            _eigenPod,
            0,
            abi.encodeWithSelector(
                IEigenPod.claimDelayedWithdrawals.selector,
                type(uint256).max
            )
        );
        account.invoke(address(this), podOwner.balance, "");

        withdrawingNodeAmount -= _nodeAmount;
    }
```

In EigenPod, it ends up calling delayedWithdrawalRouter.createDelayedWithdrawal() to create the delayed withdrawal in delayedWithdrawalRouter.

```
    function _sendETH_AsDelayedWithdrawal(address recipient, uint256 amountWei) internal {
        delayedWithdrawalRouter.createDelayedWithdrawal{value: amountWei}(podOwner, recipient);
    }
...
    function _sendETH_AsDelayedWithdrawal(address recipient, uint256 amountWei) internal {
        delayedWithdrawalRouter.createDelayedWithdrawal{value: amountWei}(podOwner, recipient);
    }
```

And this requires the user to later call DelayedWithdrawalRouter.claimDelayedWithdrawals to finalize the ETH withdrawal.

```
    function claimDelayedWithdrawals(
        uint256 maxNumberOfDelayedWithdrawalsToClaim
    ) external nonReentrant onlyWhenNotPaused(PAUSED_DELAYED_WITHDRAWAL_CLAIMS) {
        _claimDelayedWithdrawals(msg.sender, maxNumberOfDelayedWithdrawalsToClaim);
    }
```

So in EigenNativeRestakingStrategy.finalizeWithdrawingNode(), we should call DelayedWithdrawalRouter.claimDelayedWithdrawals() to finalize the ETH withdrawal.
This causes finalizeWithdrawingNode to always fail and cannot finalize the ETH withdrawal.
Yaodao: In **`EigenNativeRestakingstrategy`** , the function **`finalizewithdrawingNode()`** is used to get back the ETH from **`Eigenpod`**.

```
    function finalizeWithdrawingNode(
        uint256 _nodeAmount,
        address _eigenPod
    ) external onlyGovernance {
        address podOwner = eigenPodOwners[_eigenPod];
        require(podOwner != address(0), "EigenPod not exist");

        Account account = Account(payable(podOwner));
        account.invoke(
            _eigenPod,
            0,
            abi.encodeWithSelector(
                IEigenPod.claimDelayedWithdrawals.selector,
                type(uint256).max
            )
        );
        account.invoke(address(this), podOwner.balance, "");

        withdrawingNodeAmount -= _nodeAmount;
    }
```

The function **`claimDelayedWithdrawals()`** will be called in **`_eigenPod`**, which is the instance of **`EigenPod`**.
However, there is no function **`claimDelayedWithdrawals()`** in the newest **`EigenPod`**.
Reference: https:/github.com/Layr-Labs/eigenlayer-contracts/blob/v0.2.1-goerli-m2/src/contracts/pods/EigenPod.sol
The function **`claimDelayedWithdrawals()`** seems to exist in the contract **`DelayedWithdrawalRouter`**.

Reference: https:/github.com/Layr-Labs/eigenlayer-contracts/blob/v0.2.1-goerli-m2/src/contracts/pods/DelayedWithdrawalRouter.sol
As a result, no ETH will be get back because the call was incorrect.
biakia: In `EigenNativeRestakingStrategy`, the function `finalizeWithdrawingNode` is used to get ether back from `EigenPod`:

```
Account account = Account(payable(podOwner));
        account.invoke(
            _eigenPod,
            0,
            abi.encodeWithSelector(
                IEigenPod.claimDelayedWithdrawals.selector,
                type(uint256).max
            )
        );
```

It will call the function `claimDelayedWithdrawals` on `_eigenPod`. The `_eigenPod` is an instance of the contract `EigenPod`. However, in the latest version of the `EigenLayer`, there is no function named `claimDelayedWithdrawals` in contract `EigenPod `, you can see the code in eigenlayer's repo: https://github.com/Layr-Labs/eigenlayer-contracts/blob/v0.2.1-goerli-m2/src/contracts/pods/EigenPod.sol
The `claimDelayedWithdrawals` exits in the contract `DelayedWithdrawalRouter`: https://github.com/Layr-Labs/eigenlayer-contracts/blob/v0.2.1-goerli-m2/src/contracts/pods/DelayedWithdrawalRouter.sol#L99-L104
As a result, when calling the function `finalizeWithdrawingNode`, none ether will be sent back due to calling the function in a wrong contract.

Recommendation

thereksfour: It is recommended to call DelayedWithdrawalRouter.claimDelayedWithdrawals() in EigenNativeRestakingStrategy.finalizeWithdrawingNode()

```
    address public eigenPodManager;
    address public batchDeposit;
+   address public delayedWithdrawalRouter;

    event SetNewEigenPodManager(address olAddr, address newAddr);
    event EigenPodCreated(address owner, address eigenPod);

    constructor(
        address payable _controller,
        address _eigenPodManager,
        address _batchDeposit,
+       address _delayedWithdrawalRouter,
        string memory _name
    ) Strategy(_controller, _name) {
        eigenPodManager = _eigenPodManager;
        batchDeposit = _batchDeposit;
+       delayedWithdrawalRouter = _delayedWithdrawalRouter;
    }
...
    function finalizeWithdrawingNode(
        uint256 _nodeAmount,
        address _eigenPod
    ) external onlyGovernance {
        address podOwner = eigenPodOwners[_eigenPod];
        require(podOwner != address(0), "EigenPod not exist");

        Account account = Account(payable(podOwner));
        account.invoke(
-           _eigenPod,
+           delayedWithdrawalRouter
            0,
            abi.encodeWithSelector(
                IEigenPod.claimDelayedWithdrawals.selector,
                type(uint256).max
            )
        );
        account.invoke(address(this), podOwner.balance, "");

        withdrawingNodeAmount -= _nodeAmount;
    }
```

Yaodao: Recommend confirming the codes and updating the call of incorrect function.
biakia: Consider using the correct target in function `**finalizeWithdrawingNode**`.

Client Response

thereksfour: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8567f58cd8183640ed2009de4fec4c6b39b16385

Yaodao: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8567f58cd8183640ed2009de4fec4c6b39b16385
biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8567f58cd8183640ed2009de4fec4c6b39b16385

STO-2:Potential array out-of-bounds error

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	8olidity, biakia

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L246-L256
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L246-L256

```
246: function getEigenPods(
247:         uint256 _start,
248:         uint256 _limit
249:     ) external view returns (address[] memory pods) {
250:         uint256 length = eigenPods.length;
251:         pods = new address[](length);
252:
253:         for (uint256 i; i < _limit; i++) {
254:             pods[i] = eigenPods[_start + i];
255:         }
256:     }
```


```
246: function getEigenPods(
247:         uint256 _start,
248:         uint256 _limit
249:     ) external view returns (address[] memory pods) {
250:         uint256 length = eigenPods.length;
251:         pods = new address[](length);
252:
253:         for (uint256 i; i < _limit; i++) {
254:             pods[i] = eigenPods[_start + i];
255:         }
256:     }
```


Description

8olidity: In the getEigenPods function, there is a potential issue of accessing the array out-of-bounds when the values of _start and _limit passed as parameters result in _start + _limit being greater than the length of the eigenPods array (length). This could lead to unpredictable behavior or errors, posing a risk to the contract's security and stability.

```
    function getEigenPods(
        uint256 _start,
        uint256 _limit
    ) external view returns (address[] memory pods) {
        uint256 length = eigenPods.length;
        pods = new address[](length);

        for (uint256 i; i < _limit; i++) {
            pods[i] = eigenPods[_start + i];
        }
    }
```

biakia: In `EigenNativeRestakingStrategy`, the function `getEigenPods` will return the data from `_start` to `_start+_limit`:

```
function getEigenPods(
        uint256 _start,
        uint256 _limit
    ) external view returns (address[] memory pods) {
        uint256 length = eigenPods.length;
        pods = new address[](length);

        for (uint256 i; i < _limit; i++) {
            pods[i] = eigenPods[_start + i];
        }
    }
```

When the `length` of the `eigenPods` is less than `_start + _limit`, an array out-of-bounds error will happen.

Recommendation

8olidity: Recommended Solution: It is advisable to include necessary checks at the beginning of the function to ensure that the values of _start and _limit do not cause an out-of-bounds access. The following pseudocode can be used as a reference:

```
function getEigenPods(
    uint256 _start,
    uint256 _limit
) external view returns (address[] memory pods) {
    uint256 length = eigenPods.length;

    require(_start < length, "Start index exceeds array length");
    require(_start + _limit <= length, "End index exceeds array length");

    pods = new address[](_limit);

    for (uint256 i; i < _limit; i++) {
        pods[i] = eigenPods[_start + i];
    }
}
```

biakia: Consider adding a checking on the `_start + _limit`:

```
function getEigenPods(
        uint256 _start,
        uint256 _limit
    ) external view returns (address[] memory pods) {
        uint256 length = eigenPods.length;
        require(_start+_limit<length,"out of bounds");
        pods = new address[](length);

        for (uint256 i; i < _limit; i++) {
            pods[i] = eigenPods[_start + i];
        }
    }
```

Client Response

8olidity: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/c0bcad0d444e2b752cbb25dd844856b3308ba7fe
biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/c0bcad0d444e2b752cbb25dd844856b3308ba7fe

STO-3:Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Privilege Related	Low	Fixed	ravikiran_web3, Yaodao, biakia

Code Reference

- code/contracts/strategies/eigen/Account.sol#L26-L29
- code/contracts/strategies/eigen/Account.sol#L26-29
- code/contracts/strategies/eigen/Account.sol#L26-L29

```
26: function transferAdmin(address _admin) public {
27:         require(msg.sender == admin, "not admin");
28:         admin = _admin;
29:     }
```


```
26: function transferAdmin(address _admin) public {
27:         require(msg.sender == admin, "not admin");
28:         admin = _admin;
29:     }
```

```
26: function transferAdmin(address _admin) public {
27:         require(msg.sender == admin, "not admin");
28:         admin = _admin;
29:     }
```


Description

ravikiran_web3: The transferAdmin() function does not check for zero address. But, if the zero address is set, then the role will be lost permanently.
Also, in the case of key roles like admin, it is recommended to adopt a two step transfer to eliminate the problems arising due to human error.
Yaodao: The contract `Account` uses function `transferAdmin()` to transfer the ownership directly.

```
    function transferAdmin(address _admin) public {
        require(msg.sender == admin, "not admin");
        admin = _admin;
    }
```

It is possible that the `onlyAuth` role mistakenly transfers ownership to the wrong address, resulting in the loss of the onlyOnwer role.
biakia: The contract `Account` does not implement a two-step process for transferring ownership:

```
function transferAdmin(address _admin) public {
        require(msg.sender == admin, "not admin");
        admin = _admin;
    }
```

So ownership of the contract can be easily lost when making a mistake when transferring ownership.

Recommendation

ravikiran_web3: The recommendation is to adopt the two step transfer approach as implemented in Ownable2Step contract by Openzeppelin.
Taking an approach similar to Ownable2Step will prevent any possibility of human error. This is because the new admin will have to claim the role,
that means the account configured as new role should be accessible to the maintainers of the protocol.
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol
Yaodao: Recommend implementing a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOnwership()` function for the transfer of the ownership to fully succeed.
biakia: Consider Ownable2StepUpgradeable(https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

Client Response

ravikiran_web3: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/ca23b58bfca05c2c71171cd3199ce5c9892350cc
Yaodao: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/ca23b58bfca05c2c71171cd3199ce5c9892350cc
biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/ca23b58bfca05c2c71171cd3199ce5c9892350cc

STO-4:EigenNativeRestakingStrategy::instantWithdraw() should not have notAtSameBlock modifier to allow instant withdrawal

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ravikiran_web3

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L47-L57
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L59-L69

```
47: function withdraw(
48:         uint256 _amount
49:     )
50:         public
51:         override
52:         onlyController
53:         notAtSameBlock
54:         returns (uint256 actualAmount)
55:     {
56:         actualAmount = _withdraw(_amount);
57:     }
```

```
59: function instantWithdraw(
60:         uint256 _amount
61:     )
62:         public
63:         override
64:         onlyController
65:         notAtSameBlock
66:         returns (uint256 actualAmount)
67:     {
68:         actualAmount = _withdraw(_amount);
69:     }
```

Description

ravikiran_web3: instantWithdraw() function should allow for withdrawal in the same block unlike the withdraw() function.
But, the implementation for instantWithdraw() is same as withdraw() function and hence will render exactly same functionality of prevent withdrawal until a specified time period.
Refer to the instantWithdraw() function which also uses the **notAtSameBlock** modifier which will not allow the withdrawal in the same block.

```
function instantWithdraw(
        uint256 _amount
    )
        public
        override
        onlyController
        notAtSameBlock
        returns (uint256 actualAmount)
    {
        actualAmount = _withdraw(_amount);
    }
```

The implementation of notAtSameBlock modifier with the condition that latestUpdateTime + bufferTime should be less than block.timesamp which is updated on deposit and on last withdrawl. Hence, it functions to prevent immediate withdrawal for the last deposit.

```
   modifier notAtSameBlock() {
        require(
            latestUpdateTime + bufferTime < block.timestamp,
            "at the same block"
        );
        _;
    }
```

Recommendation

ravikiran_web3: The recommendation is to remove the **notAtSameBlock** modifier from the instantWithdraw() function

Client Response

ravikiran_web3: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/9e634b3a5de001f5102062362833c3193a0e1b83

STO-5:Variables that could be declared as immutable

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	biakia

Code Reference

- code/contracts/strategies/eigen/Account.sol#L16-L19

```
16: constructor(address _admin) {
17:         owner = msg.sender;
18:         admin = _admin;
19:     }
```

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L30-L38

```
30: constructor(
31:         address payable _controller,
32:         address _eigenPodManager,
33:         address _batchDeposit,
34:         string memory _name
35:     ) Strategy(_controller, _name) {
36:         eigenPodManager = _eigenPodManager;
37:         batchDeposit = _batchDeposit;
38:     }
```

Description

biakia: In contract `**EigenNativeRestakingStrategy**`, the variable `**batchDeposit**` assigned in the constructor can be declared as immutable.
In contract `**Account**`, the variable `**owner**` assigned in the constructor can be declared as immutable.
Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

biakia: We recommend declaring the variable `**owner**` and `**batchDeposit**` as immutable. Please note that the immutable keyword only works in Solidity version v0.6.5 and up.

Client Response

biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8eba1196477ec343285774a6c58ae4c3c5e6c88f

STO-6:Unused variables and imports

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	ravikiran_web3, biakia

Code Reference

- code/contracts/strategies/eigen/Account.sol#L14
- code/contracts/strategies/eigen/Account.sol#L14

```
14: address public eigenPod;
```


```
14: address public eigenPod;
```


- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L11

```
11: import {IBatchDeposit} from "../../interfaces/IBatchDeposit.sol";
```

Description

ravikiran_web3: eigenPod is declare in Account contract, but was never used.
biakia: In the contract `**Account**`, the variable `**eigenPod**` is never used.
In the contract `**EigenNativeRestakingStrategy**`, the following import is never used:

```
import {IBatchDeposit} from "../../interfaces/IBatchDeposit.sol";
```

Recommendation

ravikiran_web3: Revisit the eigenPod variable in Account to evaluate if the variable can be deleted from the Account Contract.
biakia: If these variables or imports are not intended to be used, it is recommended to remove them to save gas.

Client Response

ravikiran_web3: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/3062af6ae2f3917609d64acbe101aaec9f87e11f
biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/3062af6ae2f3917609d64acbe101aaec9f87e11f

STO-7:Unused function param

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	ravikiran_web3, 0xac, biakia, Yaodao

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L156-L169
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L156-L169
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L156-L169
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L156-169

```
156: function unstakeFromEigenPod(
157:         uint256 _nodeAmount,
158:         address _eigenPod
159:     ) external onlyGovernance {
160:         address podOwner = eigenPodOwners[_eigenPod];
161:         require(podOwner != address(0), "EigenPod not exist");
162:
163:         Account account = Account(payable(podOwner));
164:         account.invoke(
165:             _eigenPod,
166:             0,
167:             abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
168:         );
169:     }
```


```
156: function unstakeFromEigenPod(
157:         uint256 _nodeAmount,
158:         address _eigenPod
159:     ) external onlyGovernance {
160:         address podOwner = eigenPodOwners[_eigenPod];
161:         require(podOwner != address(0), "EigenPod not exist");
162:
163:         Account account = Account(payable(podOwner));
164:         account.invoke(
165:             _eigenPod,
166:             0,
167:             abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
168:         );
169:     }
```

```
156: function unstakeFromEigenPod(
157:         uint256 _nodeAmount,
158:         address _eigenPod
159:     ) external onlyGovernance {
160:         address podOwner = eigenPodOwners[_eigenPod];
161:         require(podOwner != address(0), "EigenPod not exist");
162:
163:         Account account = Account(payable(podOwner));
164:         account.invoke(
165:             _eigenPod,
166:             0,
167:             abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
168:         );
169:     }
```


```
156: function unstakeFromEigenPod(
157:         uint256 _nodeAmount,
158:         address _eigenPod
159:     ) external onlyGovernance {
160:         address podOwner = eigenPodOwners[_eigenPod];
161:         require(podOwner != address(0), "EigenPod not exist");
162:
163:         Account account = Account(payable(podOwner));
164:         account.invoke(
165:             _eigenPod,
166:             0,
167:             abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
168:         );
169:     }
```


Description

ravikiran_web3: unstakeFromEigenPod() function takes **_nodeAmount** as parameter along with pod address. But, internally it invokes on pod with **withdrawBeforeRestaking()** which does not accept any parameter.
Hence _nodeAmount is a redundant parameter.

```
function unstakeFromEigenPod(
    uint256 _nodeAmount,
    address _eigenPod
) external onlyGovernance {
    address podOwner = eigenPodOwners[_eigenPod];
    require(podOwner != address(0), "EigenPod not exist");

    Account account = Account(payable(podOwner));
    account.invoke(
        _eigenPod,
        0,
        abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
    );
}
```

and IEigenPod interface declaration that defines the signature of withdrawBeforeRestaking() as below.

```
interface IEigenPod {
....
function withdrawBeforeRestaking() external;
```

0xac: The `uint256 _nodeAmount` argument passed to the `unstakeFromEigenPod()` function is not used in the function and is a redundant argument.

```
function unstakeFromEigenPod(
    uint256 _nodeAmount, address _eigenPod()
    address _eigenPod
) external onlyGovernance {
    address podOwner = eigenPodOwners[_eigenPod];
    require(podOwner ! = address(0), "EigenPod not exist");

    Account account = Account(payable(podOwner));
    account.invoke(
        _eigenPod.
        0,
        abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
    );
}
```

biakia: In function `unstakeFromEigenPod`, the param `_nodeAmount` is unused.
Yaodao: The parameter `_nodeAmount` is declared but not used in the function `unstakeFromEigenPod()`.

Recommendation

ravikiran_web3: Recommendation is to remove the redundant parameter
0xac: It is suggested that the `unstakeFromEigenPod()` function be modified as follows

```
function unstakeFromEigenPod(
    address _eigenPod
) external onlyGovernance {...}
```

biakia: Consider removing the unused function param if it is not intended to be used.
Yaodao: Recommend removing the unused parameter or updating the logic.

Client Response

ravikiran_web3: Fixed.fixed: https://github.com/stakestone/stone-vault-v1/commit/00c0a6fb9b12a60677d24a137b9dbf0e4a97b571
0xac: Fixed.fixed: https://github.com/stakestone/stone-vault-v1/commit/00c0a6fb9b12a60677d24a137b9dbf0e4a97b571
biakia: Fixed.fixed: https://github.com/stakestone/stone-vault-v1/commit/00c0a6fb9b12a60677d24a137b9dbf0e4a97b571
Yaodao: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/00c0a6fb9b12a60677d24a137b9dbf0e4a97b571

STO-8:Replace abi.encodeWithSelector by abi.encodeCall in several functions in contract EigenNativeRestakingStrategy

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	ginlee

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L115
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L167
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L182
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L208
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L235

```
115: abi.encodeWithSelector(IEigenPodManager.createPod.selector)
```

```
167: abi.encodeWithSelector(IEigenPod.withdrawBeforeRestaking.selector)
```

```
182: abi.encodeWithSelector(
```

```
208: abi.encodeWithSelector(
```

```
235: abi.encodeWithSelector(
```

Description

ginlee: Since 0.8.11, abi.encodeCall provide type-safe encode utility comparing with abi.encodeWithSelector. abi.encodeWithSelector can use with interface..selector to prevent typo error, but it doesn't provide type checking. abi.encodeCall provide type checking during compile time.
For more details, please refer to link below
https://github.com/OpenZeppelin/openzeppelin-contracts/issues/3693

Recommendation

ginlee: Replace abi.encodeWithSelector by abi.encodeCall

Client Response

ginlee: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/d569296cf2d753cf61bc02b9ebcede2e8a2a06b7

STO-9:Lack of check `address(0)`

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	8olidity, biakia

Code Reference

- code/contracts/strategies/eigen/Account.sol#L16-L19

```
16: constructor(address _admin) {
17:         owner = msg.sender;
18:         admin = _admin;
19:     }
```

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L262-L269

```
262: function setNewEigenPodManager(
263:         address _eigenPodManager
264:     ) external onlyGovernance {
265:         emit SetNewEigenPodManager(eigenPodManager, _eigenPodManager);
266:
267:         eigenPodManager = _eigenPodManager;
268:     }
```

Description

8olidity: In the setNewEigenPodManager function, there is a missing check to verify whether the _eigenPodManager address is set to the zero address (address(0)). This omission may lead to unintended consequences or vulnerabilities if an invalid or zero address is passed, affecting the proper functioning and security of the contract.

```
  function setNewEigenPodManager(
        address _eigenPodManager
    ) external onlyGovernance {
        emit SetNewEigenPodManager(eigenPodManager, _eigenPodManager);

        eigenPodManager = _eigenPodManager;
    }
```

biakia: In contract `**Account**`, the constructor will not check whether the `**_admin**` is `**address(0)**`:

```
constructor(address _admin) {
        owner = msg.sender;
        admin = _admin;
    }
```

If the `**_admin**` is `**address(0)**`, the function `**transferAdmin**` will later fail to change the variable `**admin**`.

Recommendation

8olidity: It is recommended to add a validation check to ensure that the _eigenPodManager address is not set to the zero address. Here's an example pseudocode illustrating the addition of this check:

```
function setNewEigenPodManager(
    address _eigenPodManager
) external onlyGovernance {
    require(_eigenPodManager != address(0), "Invalid eigenPodManager address");

    emit SetNewEigenPodManager(eigenPodManager, _eigenPodManager);

    eigenPodManager = _eigenPodManager;
}
```

biakia: Consider adding a check in the constructor:

```
constructor(address _admin) {
        require(_admin != address(0),"invalid admin");
        owner = msg.sender;
        admin = _admin;
    }
```

Client Response

8olidity: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8cc3cf55f60ce770f0e8add80af42275aa0b3ba4
biakia: Fixed. fixed: https://github.com/stakestone/stone-vault-v1/commit/8cc3cf55f60ce770f0e8add80af42275aa0b3ba4

STO-10:Be cautious when using block.timestamp in Arbitrum

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	ginlee

Code Reference

- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L44
- code/contracts/strategies/eigen/EigenNativeRestakingStrategy.sol#L81

```
44: latestUpdateTime = block.timestamp;
```

```
81: latestUpdateTime = block.timestamp;
```

Description

ginlee: In Arbitrum, block.timestamp is similar in behavior to L1, but two different blocks in L2 can have the same block.timestamp, unlike in L1. For details, check out the link below.
https://docs.arbitrum.io/for-devs/concepts/differences-between-arbitrum-ethereum/block-numbers-and-time

Recommendation

ginlee: For scenarios requiring precise timing, consider using timestamp data provided by oracles. This approach can offer a more reliable source of time than the blockchain itself.

Client Response

ginlee: Acknowledged. Only deploy on Ethereum Mainnet

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.