Quantstamp

SSV.Network

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Payment
Timeline	2023-03-15 through 2023-03-24
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	SSV Tech Overview SSV Network Documentation Docs v3
Source Code	• bloxapp/ssv-network #16046e8
Auditors	• Jennifer Wu Auditing Engineer • Rabib Islam Auditing Engineer • Cameron Biniamow Auditing Engineer • Roman Rohleder Senior Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	25 Fixed: 6 Acknowledged: 17 Mitigated: 2
High severity findings	0
Medium severity findings	5 Fixed: 3 Acknowledged: 2
Low severity findings	13 Fixed: 3 Acknowledged: 10
Undetermined severity findings	1 Mitigated: 1
Informational findings	6 Acknowledged: 5 Mitigated: 1

Summary of Findings

The ssv.network is a decentralized protocol that facilitates the distributed operation of an Ethereum validator by employing a Distributed Validator Technology (DVT) approach. This method divides the validator key among various non-trusting nodes operated by different entities. The key is fragmented, encrypted, and distributed among these operators to execute validator tasks.

To manage their validators, the validator owners compensate their chosen operators with SSV tokens as operating fees, while paying a network fee to the DAO. The SSV token functions as a reciprocal reward system between validators and operators, and also serves as the payment token for the SSV network.

The audit focused on the smart contracts responsible for facilitating a registry for validators and operators and the distribution of SSV token payments between them. The following were identified as key areas of concern, but not limited to:

- Liquidation: This refers to the risk of clusters being liquidated incorrectly, which could lead to a loss of funds for the validators.
- Insolvency: This risk involves the possibility of clusters being insolvent, which could lead to operators being unable to withdraw.
- Accounting: The contracts must accurately account for operators' and clusters' balances when taking snapshots.
- Permission: The contracts must restrict non-permissioned operations to prevent unauthorized changes to the registry or payment distribution.
- Event data: The contracts must emit correct values in events that are used by off-chain components.

During the audit, we identified several medium-severity issues that require attention. The first issue relates to the liquidation delay, which can result in an insufficient balance of the SSV token in the `SSVNetwork` contract, preventing operators from being able to withdraw their earnings and causing the cluster to remain insolvent even upon reactivation. The second issue concerns the `withdraw()` function, which allows the cluster owner to withdraw funds from the cluster balance. However, the liquidation check is done before the withdrawal amount is subtracted from the cluster balance, making it possible for a cluster owner to withdraw funds that could cause the cluster to become liquidatable. Finally, the third issue is related to the registration of identical operators for a validator, as the `registerValidator()` function checks that the operator IDs are sorted in ascending order but does not verify whether there are any duplicate operator IDs.

Additionally, to enhance the codebase, we recommend refactoring redundant checks and documenting the code more comprehensively. For instance, the `registerValidator()` function is responsible for adding a new validator and registering a new cluster, which makes the function complex. We recommend that the client considers breaking such functions down into smaller, more specialized functions that handle specific tasks. This approach improves the code's readability, maintainability, and testability.

We recommend that the client addresses and/or considers all the findings highlighted in this report.

Fix Review: Following the initial audit, the SSV team updated the codebase to include fixes for issues and redesigned the codebase's architecture that utilizes the Diamond Storage Pattern for upgradeable module contracts. Through the use of `delegatecall`, the `SSVNetwork` contract performs state changes through the execution of code within the module contracts (`SSVClusters`, `SSVDAO`, `SSVOperators`, and `SSVViews`).

During the fix review, the client identified and addressed a critical front-running vulnerability related to validator registration. We reviewed the fix and believe that the front-running risk, affecting both validators' and operators' registrations, is still possible and the issue is documented in SSV-18. We also updated existing issues SSV-9 and SSV-13 and included additional issues SSV-19 to SSV-25 and best practices identified during the fix review.

Fix Review 2: The client resolved issues SSV-18 to SSV-25 by implementing fixes, mitigating, or acknowledging them. Although SSV-18 is acknowledged, the client provided the explanation that additional validation will be performed by the nodes off-chain to mitigate slashable events.

ID	DESCRIPTION	SEVERITY	STATUS
SSV-1	Delayed Liquidation Results in Insufficient SSV Balance	● Medium ⓘ	Acknowledged
SSV-2	Cluster Owner Can Withdraw Into Liquidation	● Medium ⓘ	Fixed
SSV-3	Register Same Operators	● Medium ⓘ	Fixed
SSV-4	Inactive Operators Can Lead to Insufficient Consensus	● Low ⓘ	Acknowledged
SSV-5	Uncapped State Variables	● Low ⓘ	Acknowledged
SSV-6	Operator Fee Cannot Increase Once Set to Zero	● Low ⓘ	Acknowledged
SSV-7	Fees Charged for Invalid Setup	● Low ⓘ	Acknowledged
SSV-8	Same Operator Registered Multiple Times	● Low ⓘ	Fixed
SSV-9	Missing Input Validation	● Low ⓘ	Acknowledged
SSV-10	`Type256.shrink()` Can Overflow	● Low ⓘ	Fixed
SSV-11	Inconsistent Use of `shrink()` and `expand()`	● Low ⓘ	Acknowledged
SSV-12	Renouncable Ownership	● Low ⓘ	Acknowledged
SSV-13	Privileged Roles	● Low ⓘ	Acknowledged
SSV-14	Keyshare Reuse Could Lead to Compromising Validator Private Key	● Informational ⓘ	Acknowledged
SSV-15	Block Timestamp Manipulation	● Informational ⓘ	Acknowledged
SSV-16	Incompatible with Deflationary Tokens	● Informational ⓘ	Acknowledged
SSV-17	Colluding Operators Can Act Maliciously on Behalf of a Validator	● Undetermined ⓘ	Mitigated
SSV-18	Front-Running Calls to `registerOperator()` and `registerValidator()`	● Medium ⓘ	Acknowledged
SSV-19	Incomplete Error Handling for Failed Delegatecall	● Medium ⓘ	Fixed
SSV-20	Same Validator Public Key Registered by Different Callers	● Low ⓘ	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
SSV-21	Fee Change Request Viewer Function Fails with Zero Fee Request	• Low ⓘ	Fixed
SSV-22	Use of Unsafe Transfer Functions	• Low ⓘ	Acknowledged
SSV-23	Application Monitoring Can Be Improved by Emitting More Events	• Informational ⓘ	Acknowledged
SSV-24	Missing Access Controls	• Informational ⓘ	Mitigated
SSV-25	Risk of Killing Upgrades	• Informational ⓘ	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Please note that the audit scope is limited to the registry and payment distribution between validators and operators. As a result, the following areas of concern are considered out of scope:

- Malicious operators: This refers to the risk of operators working together to manipulate the consensus process in their favor, which could lead to a validator being slashed for behaving dishonestly.
- Private key compromise: This risk involves the possibility of an attacker reconstructing a validator's private key from shares, which could allow them to access the validator.
- Idle validator slashing: This risk involves idle operators in the consensus process, which could result in validators losing out on block proposals and attestation rewards.
- Validator unstaking after the Shanghai fork: This risk refers to the possibility that validators may unstake their funds following the Shanghai fork, which could result in the potential incompatibility of the SSV network.
- SSV Cli key generation: The registry relies on off-chain mechanisms to handle the generation of key shares for operators.

The integration of these contracts with the remainder of the system was not subject to auditing.

If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

SSV-1
Delayed Liquidation Results in Insufficient SSV Balance

● Medium ⓘ Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> Advice taken. Implementation delayed for next release.

File(s) affected: `SSVNetwork.sol`

Description: Liquidations ensure that clusters have an adequate amount of SSV deposited to cover unclaimed network and operator fees. This is critical as operators continuously receive SSV rewards regardless of the cluster's SSV balance.

In the case where a cluster's SSV balance falls below the liquidation threshold and is not liquidated, the operators risk not being able to withdraw their earnings as the SSV balance of `SSVNetwork` is insufficient. Additionally, the cluster may remain insolvent upon reactivation due to mismatch between the operators' balances and the cluster's balance.

Exploit Scenario:
Consider a cluster with the following parameters:

- Balance: `100 SSV`
- Blocks Before Liquidation: `3`
- Validator Count: `1`
- Operators:
    - Operator ID: 1
    - Fee: `5 SSV` / block
    - Operator ID: 2
    - Fee: `2 SSV` / block
    - Operator ID: 3
    - Fee: `3 SSV` / block
    - Operator ID: 4
    - Fee: `1 SSV` / block
- Network Fee: `1 SSV` / block
- Liquidation Threshold: 36 SSV

By adding all of the operator's fees and the network fee, the cost of each block for the cluster is `12 SSV`.

1. After five blocks, the unclaimed SSV fee for the operators and network is `60 SSV`. The cluster balance is `40 SSV`, which is above the minimum liquidation threshold of `36 SSV`.
2. After the sixth block, the unclaimed SSV fee for the operators and network is `72 SSV`. The cluster balance is `28 SSV`. The cluster is now liquidatable.
3. Assume the cluster is not liquidated until the ninth block. The unclaimed SSV fee for the operators and network becomes `108 SSV`. The cluster balance is `0 SSV` since a cluster balance cannot go negative.
4. Since the difference between the unclaimed rewards and the cluster balance is negative, the liquidator does not receive any SSV reward for liquidating.
5. The SSV unclaimed rewards after liquidation are as follows:
    - Operator 1: `45 SSV`
    - Operator 2: `18 SSV`
    - Operator 3: `27 SSV`
    - Operator 4: `9 SSV`
    - Network: `9 SSV`
6. All operators withdraw their SSV rewards. Since `100 SSV` were deposited, the remaining SSV balance of the contract is `1 SSV`.
7. Withdrawing the network fees fails because the fee balance is `9 SSV` but the contract balance is only `1 SSV`.

This scenario highlights the importance of liquidations in ensuring that clusters have enough SSV deposited to cover network and operator fees. In the event that a cluster's SSV balance falls below the liquidation threshold and is not liquidated, operators may not be able to withdraw their earnings if the SSV balance of `SSVNetwork` is insufficient. This underscores the need to ensure that liquidations are promptly executed to avoid such a situation.

Recommendation: An increased liquidation threshold will reduce the chances of clusters getting liquidated too late. This can be accomplished by increasing `minimumBlocksBeforeLiquidation` or `minimumLiquidationCollateral`. Further, as the liquidation threshold increases, so does the incentive for liquidators. Furthermore, if there is a negative balance, the negative balance should be paid upon cluster reactivation to prevent contract SSV insolvency. Note if repayment of the negative balance is required upon reactivation, the current implementation does not prevent validator owners from creating a different cluster setup to bypass the repayment requirement.

SSV-2 Cluster Owner Can Withdraw Into Liquidation

● Medium ⓘ Fixed

Update
The client fixed the issue in commit `9c9206f` by validating the cluster for liquidation after deducting the amount from the cluster balance.

File(s) affected: `SSVNetwork.sol`

Description: The `withdraw()` function allows the cluster owner to withdraw funds from the cluster balance. The withdrawal is successful as long as there is sufficient balance and the cluster is not liquidatable. The `cluster.isLiquidatable()` function checks whether the cluster is eligible for liquidation based on operator fees, the current network fee, the minimum blocks before the liquidation, and the minimum liquidation collateral. The liquidation check is done before the `amount` is subtracted from the cluster `balance`. This means that it is possible for a cluster owner to withdraw funds that would cause the cluster to become liquidatable.

Exploit Scenario:
1. Alice is the owner of the cluster.
2. Alice noticed that her cluster balance is eligible for liquidation soon.
3. Alice wants to withdraw her entire balance.
4. Alice withdraws the entire cluster balance to zero using `withdraw()`.
5. It is not possible to liquidate Alice's cluster since the cluster balance is zero.

Recommendation: To prevent the possibility of withdrawing into liquidation, it is recommended to perform the liquidation check after the withdrawal amount is subtracted from the cluster balance in the `withdraw()` function. This ensures that the cluster owner cannot withdraw funds that could cause the cluster to become liquidatable, and reduces the risk of unintended consequences or vulnerabilities in the code.

SSV-3 Register Same Operators

● Medium ⓘ Fixed

Update
The client fixed the issue in commit `2c48334` by checking for duplicates in the ordered operator ids list.

File(s) affected: `SSVNetwork.sol`

Description: While the `registerValidator()` function validates the operator count using `_validateOperatorIds()`, it does not prevent the registration of identical operators for a validator. The function only checks that the operator IDs are sorted in ascending order but does not verify whether there are any duplicate operator IDs. If duplicate operator IDs are registered, it could lead to issues with consensus among the operators.

Recommendation: Validate the uniqueness of each operator ID when registering a new validator set up in the function `registerValidator()`.

SSV-4 Inactive Operators Can Lead to Insufficient Consensus

● Low ⓘ Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> It's validators' owners responsibility to track operators' performance (via external tools, etc)

File(s) affected: `SSVNetwork.sol`

Description: The `SSVNetwork.removeOperator()` function allows operators to remove themselves from the clusters they were previously operating for. Although this action causes the operators to stop receiving SSV rewards, the clusters are not updated to reflect the operator's removal. As a result, if sufficiently many operators are removed, consensus may not be reached.

In addition, if an operator stops running an operator node without calling `removeOperator()`, they will no longer contribute to the consensus threshold, but they will still receive SSV rewards from the cluster owner. Based on the current implementation, the validator still owes payments to any active operators even though the setup is invalid. Therefore, validators must track the performance of the operators they use to ensure the success of their validator and maximize their rewards.

Exploit Scenario:
1. A validator registers a cluster using `registerValidator()` with the operator IDs: `1`, `2`, `3`, and `4`.

2. Operator `2` calls `removeOperator()`.
3. The other operators are still able to act on behalf of the validator since only three operators are needed to reach the consensus threshold.
4. Operator `4` calls `removeOperator()`.
5. Since only two operators are active, the consensus threshold is not met.
6. The validator is now unable to propose or attest blocks and will miss out on validator rewards.

Recommendation: It is recommended to update the documentation to provide clear guidelines for validator owners regarding the management of their operator setup. Validator owners should be informed that they are responsible for tracking the performance of the operators they use to ensure the success of their validator and maximize their rewards. To maintain a sufficient number of active operators and avoid any issues with consensus or loss of rewards, validator owners should regularly monitor their operator setup for any changes or issues. This may include tracking operators that have been removed or become inactive, as well as identifying operators that are not stable or have poor performance. Validator owners should also be aware of any changes in operator fees that may affect the profitability of their validator.

SSV-5 Uncapped State Variables

• Low Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> Part of these variables will be updated only via upgrade process, so we could set that limits in the upgrade contracts.

File(s) affected: `SSVNetwork.sol`

Description: There are numerous state variables within `SSVNetwork` that do not have a lower or upper bound limit. While the functions that update these state variables are only accessible to the owner, an improper configuration could lead to a negative user experience or loss of funds.

1. `operatorMaxFeeIncrease`: No upper bound limit. If set to a large value, operators could increase their fee substantially after `declareOperatorFeePeriod` has passed. Unaware cluster owners are then at risk of paying high operator fees or liquidation.
2. `declareOperatorFeePeriod`: No lower bound limit. If set to a small value, operators could update their fee before cluster owners have enough time to respond.
3. `executeOperatorFeePeriod`: No lower bound limit. If set to a small value, operators would not have an adequate amount of time to execute their fee update.
4. `minimumBlocksBeforeLiquidation`: No upper bound limit. If set to a large value, all clusters are at risk for liquidation.
5. `minimumLiquidationCollateral`: No upper bound limit. If set to a large value, all clusters are at risk for liquidation.

Recommendation: Add checks for relevant lower or upper bound limits when assigning value to the aforementioned state variables.

SSV-6 Operator Fee Cannot Increase Once Set to Zero

• Low Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> This is the expected behaviour.

File(s) affected: `SSVNetwork.sol`

Description: The `_declareOperatorFee()` function calculates the maximum allowed fee increase based on the current `operatorFee` and the maximum allowed fee increase percentage `operatorMaxFeeIncrease`. If the new fee is greater than the maximum allowed fee increase, it reverts with a `FeeExceedsIncreaseLimit` error.

The operator fee can be set to zero in `declareOperatorFee()` or reduced to zero with `reduceOperatorFee()` and due to `operatorMaxFeeIncrease`, increasing the operator fee will become impossible once the operator fee is set to zero.

Recommendation: Consider ensuring that the operator fee is always greater than `MINIMAL_OPERATOR_FEE` and writing a test to ensure that the operator fee is never set to zero. If the operator fee can be set to zero, consider updating the function to allow increases from zero.

SSV-7 Fees Charged for Invalid Setup

• Low Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> Related to SSV-4

File(s) affected: `SSVNetwork.sol`

Description: If there are enough operators removed for a validator, the consensus between operators may be affected for the particular validator. For example, if there are four operators, and two operators are removed, consensus cannot be reached among the operators. The validator will become idle and lose out on block proposal and attestation rewards. Based on the current implementation, the validator still owes payments to any active operators even though the setup is invalid.

Recommendation: It is recommended to update the documentation to provide clear guidelines for validator owners regarding the management of their operator setup. Validator owners should be informed that they are responsible for tracking the performance of the operators they use to ensure the success of their validator and maximize their rewards. To maintain a sufficient number of active operators and avoid any issues with consensus or loss of rewards, validator owners should regularly monitor their operator setup for any changes or issues. This may include tracking operators that have been removed or become inactive, as well as identifying operators that are not stable or have poor performance. Validator owners should also be aware of any changes in operator fees that may affect the profitability of their validator.

SSV-8 Same Operator Registered Multiple Times

• Low ⓘ Fixed

Update
The client fixed the issue in commit `77e7093` by validating the owner of an operator's public key.

File(s) affected: `SSVNetwork.sol`

Description: Using `SSVNetwork.registerOperator()`, the same operator can register their public key multiple times where each registration generates a new operator ID. If an operator desires to exceed the `validatorsPerOperatorLimit`, the operator could register the same public key again and partner with more validators.

Recommendation: Confirm whether this is intended behavior. If not, create a mapping that sets a flag for registered operator public keys.

SSV-9 Missing Input Validation

• Low ⓘ Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

> We understand the missing input validations don't represent a major business / security risk so we keep code as it is.

File(s) affected: `SSVNetwork.sol`, `ProtocolLib.sol`, `SSVNetworkViews.sol`, `SSVDAO.sol`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.

The following is the list of places that can potentially benefit from a stricter input validation:
1. `SSVNetwork.sol#L110`: the `minimumBlocksBeforeLiquidation_` of the `__SSVNetwork_init_unchained()` can be less than `MINIMAL_LIQUIDATION_THRESHOLD`.
2. `SSVNetwork.sol#L173`: the `recipientAddress` of the `setFeeRecipientAddress()` should not be zero address.
3. `SSVNetwork.sol#L180`: the `keccak256` of `publicKey` of the `registerValidator()` should not be zero bytes.
4. `SSVNetwork.sol#L180`: the length of `shares` of the `registerValidator()` should match the length of `operatorIds`.
5. `SSVNetwork.sol#L604`: the `amount` of `withdrawNetworkEarnings()` should be greater than zero.
6. **(Fix Review)** `ProtocolLib.sol#18`: the `fee` of the `updateNetworkFee()` function not checked to be non-zero or otherwise reasonably bound.
7. **(Fix Review)** `ProtocolLib.sol#L38`: the `deltaValidatorCount` of the `updateDAO()` function is not checked to be non-zero or otherwise reasonably bound.
8. **(Fix Review)** `SSVNetworkViews.sol#L58:` the `operatorId` of the `getOperatorById()` function is not checked to be an existing operator.
9. **(Fix Review)** `SSVDAO.sol#L45`: the `percentage` of the `updateOperatorFeeIncreaseLimit()` function is not checked to be non-zero or smaller than `PRECISION_FACTOR`.

Recommendation: Add the validations and checks listed in the description.

SSV-10 `Type256.shrink()` Can Overflow

• Low ⓘ Fixed

Update
The client fixed the issue in commit `50817b0` by validating the value before shrinking.

File(s) affected: `Type256.sol`

Description: `Type256.shrink()` converts 256-bit values to 64-bit values by dividing a 256-bit value by `10,000,000` and typecasting it to a 64-bit value. Additionally, the 256-bit value must be divisible by `10,000,000` or else the function will revert.

If the input 256-bit value is greater than `2^64 * 10,000,000`, the returned 64-bit value is incorrect.

Exploit Scenario: Consider the following values which will be impacted by an invalid shrink operation:

- `184467440737095516150000000.shrink() ==> 18446744073709551615`
- `184467440737095516160000000.shrink() ==> 0`
- `184467440737095516170000000.shrink() ==> 1`

Recommendation: Check that the input 256-bit value is less than or equal to `2^64 * 10,000,000`.

SSV-11 Inconsistent Use of `shrink()` and `expand()`

• Low ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Generally we manage inputs and event outputs in uin256 and internally use uin64, so its an expected behaviour. Added natspec comments.

File(s) affected: `SSVNetwork.sol`, `SSVNetworkViews.sol`

Description: The current implementation of the `updateNetworkFee()` function in the code snippet is emitting the entire network `fee` in the `NetworkFeeUpdated` event, despite the shrunken `fee` being saved in the `network` struct. This could lead to confusion and misinterpretation of the actual network fee.

```
        Network memory network_ = network;

        DAO memory dao_ = dao;
        dao_.updateDAOEarnings(network.networkFee);
        dao = dao_;

        network_.networkFeeIndex = NetworkLib.currentNetworkFeeIndex(network_);
        network_.networkFeeIndexBlockNumber = uint64(block.number);

        emit NetworkFeeUpdated(network_.networkFee.expand(), fee);

        network_.networkFee = fee.shrink();
        network = network_;
```

Inconsistent uses of shrunken fees were found in the following functions:

1. `fee` in `SSVNetwork.declareOperatorFee()`
2. `fee` in `SSVNetwork.reduceOperatorFee()`
3. `fee` in `SSVNetwork.registerOperator()`
4. `fee` in `SSVNetwork.updateNetworkFee()`
5. `amount` in `SSVNetwork.updateMinimumLiquidationCollateral()`
6. `fee` in `SSVNetwork.declareOperatorFee()`
7. `fee` in `SSVNetwork.reduceOperatorFee()`

Recommendation: It is recommended to update the `updateNetworkFee` function to emit the shrunken `fee` value from the `Network` struct in the `NetworkFeeUpdated` event to avoid any confusion or misinterpretation of the actual network fee.

SSV-12 Renouncable Ownership

• Low ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Will update docs to warn about this topic.

File(s) affected: `SSVNetwork.sol`, `SSVNetworkViews.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function protected by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Consider whether ownership revocation is a necessary feature. If it is not, override the ownership revocation functionality to disable it. If it is, add end-user documentation stating the risk.

SSV-13 Privileged Roles

● Low ⓘ Acknowledged

Update

The client has acknowledged the issue.

File(s) affected: `SSVNetwork.sol`, `SSVNetworkViews.sol`

Description: All privileged permissions should be clearly documented for users. The privileged permissions of each contract are documented below.

`SSVNetwork` and `SSVNetworkViews` are upgradeable contracts that can have their implementation contracts swapped out at any time through `upgradeTo()` function by the contract owner.

- `SSVNetwork.removeValidator()` can only be called by `_validatorPKs[hashedValidator].owner`
- `SSVNetwork.liquidate()` can be called by a cluster's owner when `isLiquidatable()` returns false

The following are all cases of owner privileges:

- `SSVNetwork.updateNetworkFee()`
- `SSVNetwork.withdrawNetworkEarnings()`
- `SSVNetwork.updateOperatorFeeIncreaseLimit()`
- `SSVNetwork.updateDeclareOperatorFeePeriod()`
- `SSVNetwork.updateExecuteOperatorFeePeriod()`
- `SSVNetwork.updateLiquidationThresholdPeriod()`
- `SSVNetwork.updateMinimumLiquidationCollateral()`
- **(Fix Review)** `SSVNetwork.setRegisterAuth()`: If a validator is authorized, calls `SSVNetwork.registerValidator()`, then has their authorization revoked, the validator is still active and can call `SSVNetwork.reactivate()` if the cluster becomes inactive.
- **(Fix Review)** `SSVNetwork.upgradeModule()`: This privileged function has a high exploit risk if the `owner` is compromised. If a module is upgraded to a malicious contract, the malicious contract could drain the `SSVNetwork` contract of all funds.

Recommendation: Clarify the impact of these privileged actions on the end-users via publicly facing documentation.

SSV-14 Keyshare Reuse Could Lead to Compromising Validator Private Key

● Informational ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> It's not possible to remove an operator from a cluster with the current implementation. A cluster is formed by a set of operators (4, 7, 10, or 13) and an owner. Once formed, there is no way to add/remove operators to it.

File(s) affected: `SSVNetwork.sol`

Description: The project uses Shamir Secret Sharing to split validator private keys into encrypted key shares for distribution to operators. However, suppose an operator is removed from a cluster and later added back in. The case where the private key would be split into the same key shares as in the initial configuration and a different encrypted key share would be sent to an operator should be avoided, as this may result in an operator potentially accumulating the quorum number of key shares, compromising the validator private key.

Recommendation: Ensure that in a given operator configuration, an operator will only ever have access to one distinct keyshare.

SSV-15 Block Timestamp Manipulation

● Informational ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> We have evaluated the risks and will continue using block.timestamp.

File(s) affected: `SSVNetwork.sol`

Description: At `SSVNetwork.sol#770`, comparisons are made against `block.timestamp`. Projects may rely on block timestamps for various purposes. Given that this project may, in principle, be deployed on any EVM chain, it is important to note that different networks may impose different requirements for their validators when it comes to setting the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

Recommendation: Consider the degree to which validators may influence `block.timestamp` on the target networks.

SSV-16 Incompatible with Deflationary Tokens

● Informational ⓘ Acknowledged

> **Update**
> The client acknowledged the issue.

File(s) affected: `SSVNetwork.sol`

Description: The contracts are not designed to accommodate deflationary tokens when accepting token transfers. If a user transfers funds that incur a fee on transfer during `deposit()` or `reactivate()`, the contract will report that the user has deposited more funds than they actually sent. This can cause issues during withdrawal. This issue is provided as informational if payments are updated to accept other tokens.

Recommendation: Only account for the difference between the pre-transfer balance and the post-transfer balance of the contract. The difference will capture the amount of token that has been transferred regardless of the token transfer mechanism. Please be cautious of reentrancy if this implementation is considered.

SSV-17
Colluding Operators Can Act Maliciously on Behalf of a Validator

● Undetermined ⓘ Mitigated

> **Update**
> The issue is mitigated by adding an authorization process for operators and validators, and the SSV-3 fix prevents a validator from registering the same operator more than once. However, the risk of operators colluding within a cluster is still possible.

File(s) affected: `SSVNetwork.sol`

Description: Validators split their validator private keys into key shares using Shamir Secret Sharing and encrypt them for use by a set of operators. These operators use the key shares to propose blocks or make attestations on behalf of the validator. However, if an individual or group of operators possesses the "quorum" number of key shares, they can recover the validator's private key and make unilateral decisions that may lead to slashing.

It is important to note that an operator may obtain multiple key shares in two ways: by registering as an operator multiple times when setting up a new cluster or by colluding with others who have registered as operators and sharing key shares with one another. If a cluster contains colluding operators, they can act maliciously and cause a slashing event by using their key shares to make decisions on behalf of the validator.

To prevent such malicious activity, validators should carefully select trusted, high-performing, and decentralized operators for their clusters. It is critical to communicate this risk to validators and emphasize the importance of proper operator selection and monitoring. It is also important to note that Ethereum 2 validators have a signing key and a withdrawal key, and only the signing key is provided to operators, which prevents colluding operators from withdrawing the validator's staked ETH.

Recommendation: There are a few ways to mitigate the possibility of operator collusion:
1. Implement KYC process:
   1. The owner of `SSVNetwork` could require addresses to undergo a KYC process before they are able to register as operators;
   2. The front end that allows validators to choose operators for inclusion in clusters could highlight the operators who have chosen to undergo a KYC process.
2. The contract could prevent validators from being registered to clusters with the same operator included more than once.

SSV-18
Front-Running Calls to `registerOperator()` and `registerValidator()`

● Medium ⓘ Acknowledged

> **Update**
> The client acknowledged the issue for both validator and operator registration. The client stated that the `sharesData` argument for the function `SSVClusters.registerValidator()` acts as a signed message and will be validated off-chain. However, this additional validation does not apply to `SSVOperators.registerOperator()` as there is no `sharesData` argument; but if the issue occurs, the operators can re-register. The client provided the following explanation:
>
> > There is sort of a signed message in the validator registration included in the shares function parameter. An attacker can front-run the transaction but it won't work because the nodes won't care about it, they will drop it. What we care about most here is the slashable event and this is mitigated. For operators, if it happens to one of your operators, just make a new operator pub / private key and re-register, you have no clients yet to have to worry about trying to migrate them.

File(s) affected: `SSVNetwork.sol`, `SSVClusters.sol`, `SSVOperators.sol`

Description: When an operator calls `registerOperator()`, the operator's `publicKey` and `fee` amounts are passed as arguments. Once `registerOperator()` is called, the operator is registered with its `publicKey`, and the operator owner is set as the caller. Since the operator's `publicKey` refers to the operator's SSV node and differs from the caller's public key, there is no way of verifying that the caller is the actual owner of the operator's `publicKey`. Therefore, a malicious user could see a `registerOperator()` call in the pending transaction pool and front-run the transaction with the operator's `publicKey`. The same vulnerability is present with validators when calling `registerValidator()`.

While the caller of `registerOperator()` or `registerValidator()` needs prior authorization from the `owner` via the `SSVNetwork.setRegisterAuth()` function, any registered operator or validator can front-run calls to `registerOperator()` or `registerValidator()` that are from other authorized users.

Recommendation: Consider utilizing signed messages for operator and validator registration. Verify that the caller is the owner of the operator or validator `publicKey` off-chain and supply a signed message that includes the operator/validator public key and the owner's address.

SSV-19 Incomplete Error Handling for Failed Delegatecall

● Medium ⓘ Fixed

Update
The client fixed the issue in commit `abaae50`. The `CoreLib.delegateCall()` function has been refactored into the `SSVProxy._delegate()` function which is inherited by `SSVNetwork`. The `_delegate()` function is updated to revert on error.

File(s) affected: `CoreLib.sol`

Description: The `CoreLib.delegateCall()` function uses a low-level `delegatecall` to execute the logic of another contract. When the `delegatecall` is unsuccessful, the function checks for returned data, and if present, it reverts the transaction and returns this data as an error message. However, in scenarios where the `delegatecall` fails but does not provide any return data, the function does not revert and continues to execute. This could potentially lead to undesired outcomes and unexpected behaviors because the function does not stop despite the `delegatecall` failure.

Recommendation: The `delegateCall` function should be modified to account for cases where the `delegatecall` fails, regardless of whether any return data is provided.

SSV-20 Same Validator Public Key Registered by Different Callers

● Low ⓘ Acknowledged

Update
The client acknowledged the issue and stated the `sharesData` argument for the function `SSVClusters.registerValidator()` acts as a signed message and will be validated off-chain.

File(s) affected: `SSVNetwork.sol`, `SSVClusters.sol`

Description: When registering a validator, the validator's `publicKey` and the caller's address are hashed to create a validator ID, which is considered the unique identifier for the validator and is used as the key in the `validatorPKs` mapping. Since the `registerValidator()` function does not verify that a validator's `publicKey` is unique when registering, multiple different users can call `registerValidator()` with the same `publicKey` but have unique validator identifiers.

Recommendation: Confirm if this is intended behavior. If not, consider only hashing the `publicKey` to generate the identifier for the validator.

SSV-21 Fee Change Request Viewer Function Fails with Zero Fee Request

● Low ⓘ Fixed

Update
The client fixed the issue in commit `fa55e03` by removing the check for zero fees and adding an additional return parameter, indicating if the fees have been declared, based on a non-zero approval begin time.

File(s) affected: `SSVViews.sol`, `SSVOperators.sol`

Description: The `getOperatorDeclaredFee()` function, which retrieves the details of an operator's fee change request, does not function correctly when the requested fee is zero.

The function contains a check that reverts the transaction if the requested fee is zero, returning an error of `NoFeeDeclared()` . However, setting a fee request to zero is a valid operation that an operator might wish to perform. This means that a legitimate fee change request to zero is inaccessible via this viewer function.

Recommendation: Consider modifying the `getOperatorDeclaredFee()` function to handle zero fee requests properly. This could involve removing the check for zero fees or adjusting the condition to allow retrieval of fee requests where the fee is explicitly set to zero. This would ensure all valid fee change requests, regardless of their value, are accessible through this viewer function.

SSV-22 Use of Unsafe Transfer Functions

• Low ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> We use only SSV token. In the case we manage other tokens in the future, need to check it its necessary to use a wrapper.

File(s) affected: `CoreLib.sol`

Description: Functions `CoreLib.transferBalance()` and `CoreLib.deposit()` use the ERC20 functions `transfer()` and `transferFrom()` to transfer funds from the underlying token (`SSVStorage.token`). However, there exist tokens that do not return a value and would be handled wrongly in the given context.

Recommendation: We recommend the use of a safe wrapper library that can handle the different types of ERC20 return-value behaviours.

SSV-23 Application Monitoring Can Be Improved by Emitting More Events

• Informational ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Advice taken. We will consider adding more relevant events taking into account the extra gas increase.

File(s) affected: `OperatorLib.sol` , `ProtocolLib.sol` , `ClusterLib.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs, or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:
1. `OperatorLib.updateSnapshotSt()` : `operator.snapshot.*` .
2. `OperatorLib.updateOperators()` : `operator.validatorCount` .
3. `ProtocolLib.updateNetworkFee()` : `StorageProtocol.networkFeeIndex` , `StorageProtocol.networkFeeIndexBlockNumber` and `StorageProtocol.networkFee` .
4. `ProtocolLib.updateDAOEarnings()` : `StorageProtocol.daoBalance` and `StorageProtocol.daoIndexBlockNumber` .
5. `ProtocolLib.updateDAO()` : `StorageProtocol.daoValidatorCount` .
6. `ClusterLib.updateBalance()` : `cluster.balance` .
7. `ClusterLib.updateClusterData()` : `cluster.index` and `cluster.networkFeeIndex` .

Recommendation: Consider emitting the events.

SSV-24 Missing Access Controls

• Informational ⓘ Mitigated

Update

The client mitigated the issue in commit `5d99ab1` by updating the documentation in `./docs/architecture.md` to mention that while direct contract calls are possible they do not change the main state and calls should rather be done through contracts `SSVNetwork` or `SSVNetworkViews` .

File(s) affected: `SSVClusters.sol` , `SSVDAO.sol` , `SSVOperators.sol`

Description: All externally callable functions in the following contracts are callable by anyone as they do not employ any kind of access controls:

1. `SSVClusters.sol` .
2. `SSVDAO.sol` .
3. `SSVOperators.sol` .

Direct calls to these functions circumvent the access controls performed in contract `SSVNetwork.sol`. However, performing direct calls will lead to uncontrolled storage reads/writes (`SSVStorage.load()` and `SSVStorageProtocol.load()` will be relative to the called contract and not `SSVNetwork.sol`, which holds the correct global state) and no path of exploitation was found during the audit.

Recommendation: Consider adding corresponding access controls, making sure only the correspondingly authenticated users may call these functions.

SSV-25 Risk of Killing Upgrades

● Informational Acknowledged

Update
The client acknowledged the issue.

File(s) affected: `SSVNetwork.sol`, `SSVNetworkViews.sol`

Description: The UUPS pattern was chosen for the upgradeable contracts. One of the drawbacks of using such a pattern is that if a future implementation does not implement the `upgradeTo()` function, then upgrades for the contracts have effectively been killed.

Recommendation: Understand the drawbacks of using a UUPS pattern and document the potential risks for users.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Functions should have a brief description of their purpose, input parameters, and return values (if any). The absence of documentation for functions can lead to decreased maintainability and an increased likelihood of human error when making changes to the code. Add NatSpec (see: doc) code document for all public and external functions
2. Fixed Ensure that all contracts that have privileged roles, such as `owner` or `operator owner`, have clear and documented expectations for the responsibilities and setup of those roles.
3. Fixed The documentation provides incorrect information for adding the first validator in a new cluster. It suggests using `{0,0,0,0,0,false}` instead of the correct format `{0,0,0,0,0,true}`. Using `false` as input will cause revert.
4. Fixed Remove the TODO at `SSVNetwork.sol#49`.
5. The liquidation threshold period of `SSVDAO.MINIMAL_LIQUIDATION_THRESHOLD` (`100_800`) is not mentioned in the specification.
6. The operator constraints (`MIN_OPERATORS_LENGTH = 4`, `MAX_OPERATORS_LENGTH = 13` and number of operators must be divisible by `4`) put on validators when registering are not mentioned in the specification.

Adherence to Best Practices

1. Fixed Invoke `_disableInitializers()` on the constructor of upgradeable contracts (OpenZeppelin Doc) to avoid leaving the implementation contract uninitialized.
2. Fixed The visibility of the following state vairables should be set:
   - Fixed `SSVNetworkViews._ssvNetwork`
   - Fixed `SSVNetworkViews.__gap`
   - **(Deprecated)** `SSVNetwork.__gap`
3. Fixed The cluster update in the `reactivate()` function seems to be redundant at `SSVNetwork.sol#461`. Considering removing the update to reduce gas usage.

4. Fixed `SSVNetwork.removeValidator()` validates the operator's length and validator public key with calls to `_validateOperatorIds()` and `_validatePublicKey()`. These checks are not needed since they were already performed when the cluster was registered.
5. Unresolved For further gas optimization, consider using `unchecked` blocks around arithmetic that you are certain will not overflow (for example, `++cluster.validatorCount` or `++dao_.validatorCount`).
6. Fixed `ClusterLib.validateHashedCluster()` calls `ssvNetwork.clusters()` twice to obtain the same hashed cluster data. Store the result of `ssvNetwork.clusters()` in a memory variable to avoid loading the same data from storage.
7. Fixed `SSVNetwork.registerValidator()` calls `operatorsWhitelist` twice to obtain the same whitelist address for a given operator. Consider storing the returned value in a memory variable and reference the memory variable instead.
8. Fixed `SSVNetwork._withdrawOperatorEarnings()` calls `amount.shrink()` twice to obtain the same value. Consider storing the returned value in a memory variable and reference the memory variable instead.
9. **(Deprecated)** `SSVNetwork.withdraw()` change `operator` from `storage` to `memory`.
10. Fixed In order to save storage slots in `struct Cluster`, consider ordering the fields such that `uint256 balance` occurs after all the other fields. This will ensure that only two storage slots are used, whereas three storage slots are currently being used.
11. Fixed The following variables can be renamed to accurately convey their functionality:
    1. Fixed `event OperatorFeeCancellationDeclared` should be renamed `event OperatorFeeDeclarationCancelled`
    2. Fixed `error NoFeeDelcared()` is misspelled and should be corrected `NoFeeDeclared()`.
    3. Fixed `uint64 currrentNetworkFeeIndex` should be corrected `uint64 currentNetworkFeeIndex`.
    4. Fixed `function _validateOperatorIds()` should be renamed `function _validateOperatorsLength()`
12. Fixed The local variable `owner` shadows the function `OwnableUpgradeable.owner()`; consider renaming the variable to avoid shadow conflicts in the following functions:
    1. Fixed `SSVClusters.liquidate()`
    2. Fixed `SSVClusters.deposit()`
    3. Fixed `SSVNetworkViews.getValidator()`
    4. Fixed `SSVNetworkViews.isLiquidatable()`
    5. Fixed `SSVNetworkViews.isLiquidated()`
    6. Fixed `SSVNetworkViews.getBurnRate()`
    7. Fixed `SSVNetworkViews.getBalance()`
13. Consider replacing the usage of magic numbers with constants variable.
    1. **(Deprecated)** `SSVNetwork.__SSVNetwork_init_unchained()`: `2_000`
    2. Fixed `SSVNetwork._declareOperatorFee()`: `10000`
14. Unresolved The `registerValidator()` function allows the addition of a new validator and the registration of a new cluster. This coupling of concerns makes the function complex. Consider breaking down the function into smaller, more specialized functions that handle specific tasks. This can help improve code readability, maintainability, and testability.
15. Fixed There are two functions in `SSVNetwork` with the name `withdrawOperatorEarnings()`. One allows the operator to withdraw a given amount while the other withdrawals all of the operator's earnings. For improved readability, rename to `withdrawAllOperatorEarnings()`.
16. Fixed `ISSVNetwork.registerValidator.sharesEncrypted` does not match `SSVNetwork.registerValidator.shares`. These parameter names should be made to match.
17. Fixed When using the Diamond Pattern, the gas consumption can be minimized by load storage pointers only once per function execution. The following functions access the same storage pointer more than once during a single function execution:
    1. Fixed `SSVOperators.declareOperatorFee()`
    2. Fixed `SSVDAO.updateNetworkFee()`
18. Fixed In `SSVNetworkViews`, the state variable `__gap` is missing a space between `private` and `__gap`.
19. Fixed Set the visibility of `RegisterAuth.SSV_STORAGE_POSITION` to `private`.
20. Fixed Set the visibility of `SSVStorage.SSV_STORAGE_POSITION` to `private`.
21. Fixed Set the visibility of `SSVStorageProtocol.SSV_STORAGE_POSITION` to `private`.
22. Fixed In `SSVClusters.registerValidator()`, the following `if` statement loads the data from `s.operatorsWhitelist[operatorId]` twice. Load the data into a memory variable and reference the memory variable instead.
23. Unresolved Rename `SSVOperators.MINIMAL_OPERATOR_FEE` to `MAX_OPERATOR_FEE` for improved clarity.
24. Fixed In `SSVOperators.cancelDeclareOperatorFee()`, `SSVStorage.load()` is called three times. Load the returned `StorageData` struct into a storage variable and reference the storage variable instead.
25. Fixed Remove the function `setFeeRecipientAddress()` from `SSVOperators` since the same function is already in `SSVNetwork`.
26. Remove the following unused imports:
    1. Fixed `ClusterLib.sol`
        - Remove: `import "./SSVStorageProtocol.sol";`.
    2. `ProtocolLib.sol`
        - Fixed Remove: `import "../SSVNetwork.sol";`.
        - Note: `Types.sol` is already imported via `SSVStorageProtocol.sol`.
        - Fixed Add: `import "./Types.sol";`.
    3. Fixed `RegisterAuth.sol`
        - Fixed Remove: `import "../interfaces/ISSVNetworkCore.sol";`,
        - Fixed Remove: `import "./Types.sol";`,
        - Fixed Remove: `import "@openzeppelin/contracts/utils/Counters.sol";`.
    4. Fixed `SSVStorage.sol`
        - Fixed Remove: `import "./Types.sol";`.
    5. Fixed `SSVStorageProtocol.sol`
        - Fixed Remove: `import "../interfaces/ISSVNetworkCore.sol";`.
    6. Fixed `SSVClusters.sol`
        - Fixed Remove: `import "./Types.sol";`.
    7. Fixed `SSVDAO.sol`
        - Fixed Remove: `import "../libraries/OperatorLib.sol";`.

        - Unresolved Replace: `import "../libraries/SSVStorage.sol";` with `import "../libraries/SSVStorageProtocol.sol";`.
    8. Fixed `SSVNetwork.sol`
        - Fixed Remove: `import "./libraries/OperatorLib.sol";`,
        - Fixed Remove: `import "./libraries/ClusterLib.sol";`,
        - Unresolved Remove: `import {SSVModules} from "./libraries/SSVStorage.sol";`.
        - Fixed Remove: `using ClusterLib for Cluster;`.
    9. Unresolved `SSVNetworkViews.sol`
        - Unresolved Remove: `import "./libraries/Types.sol";`,
        - Unresolved Remove: `import "./libraries/ClusterLib.sol";`,
        - Unresolved Remove: `import "./libraries/OperatorLib.sol";`,
        - Unresolved Remove: `import "./libraries/ProtocolLib.sol";`.
        - Unresolved Remove: `using Types256 for uint256;`,
        - Unresolved Remove: `using Types64 for uint64;`,
        - Unresolved Remove `using ClusterLib for Cluster;`,
        - Unresolved Remove `using OperatorLib for Operator;`.
27. Mitigated To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `CoreLib.ModuleUpgraded()`
28. Unresolved `ProtocolLib.sol#L42`: The check `> type(uint32).max` is unnecessary, as since solidity version 0.8.0 implicit overflow-/underflow-checks are performed on arithmetic.
29. Unresolved To improve readability and maintainability it is advised to use descriptive names for code elements (function names, variable names, ...). In this regard, consider the following cases:
    1. Unresolved `ProtocolLib.networkTotalEarnings()`: Consider renaming the function to i.e. `networkDAOEarnings()`, as it updates the `daoBalance` and does not take into account the other network fee (`networkFeeIndex`).
    2. Unresolved `SSVViews.getValidator()`: Consider renaming it to i.e. `isValidatorActive()`, as it more precisely represents the function.
30. Fixed It is recommended to use explicit type widths (i.e. `uint256`) over implicit ones. In this regard, consider the following instances:
    1. Fixed `SSVCluster.sol#L32`.
    2. Fixed `SSVCluster.sol#L80`.
    3. Fixed `SSVCluster.sol#L304`.
    4. Fixed `SSVCluster.sol#L305`.
    5. Fixed `SSVViews.sol#L85`.
    6. Fixed `SSVViews.sol#L86`.
    7. Fixed `SSVViews.sol#L121`.
    8. Fixed `SSVViews.sol#L122`.
    9. Fixed `SSVViews.sol#L154`.
    10. Fixed `SSVViews.sol#L155`.
    11. Fixed `OperatorLib.sol#L39`.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `d0e...d7b ./contracts/SSVNetwork.sol`
- `2d1...7ad ./contracts/ISSVNetworkViews.sol`
- `d00...2c0 ./contracts/ISSVNetwork.sol`
- `05f...bcd ./contracts/SSVNetworkViews.sol`
- `ce1...108 ./contracts/ISSVNetworkCore.sol`
- `956...7e6 ./contracts/libraries/NetworkLib.sol`
- `610...a65 ./contracts/libraries/OperatorLib.sol`
- `3ce...de1 ./contracts/libraries/ClusterLib.sol`
- `b88...0e3 ./contracts/libraries/Types.sol`

Tests

- `a89...0d7 ./test/helpers/utils.ts`
- `258...629 ./test/helpers/gas-usage.ts`
- `ec3...e5f ./test/helpers/contract-helpers.ts`
- `302...76b ./test/account/deposit.ts`
- `0f1...0ef ./test/account/withdraw.ts`

- `549...51a ./test/validators/remove.ts`
- `5bd...8d1 ./test/validators/register.ts`
- `71a...065 ./test/deployment/version.ts`
- `2ea...000 ./test/deployment/deploy.ts`
- `ba9...6e3 ./test/sanity/balances.ts`
- `f57...6ea ./test/liquidate/liquidate.ts`
- `1e5...86d ./test/liquidate/liquidated-cluster.ts`
- `e93...67e ./test/liquidate/reactivate.ts`
- `436...ae3 ./test/dao/liquidation-collateral.ts`
- `fa6...9b5 ./test/dao/liquidation-threshold.ts`
- `c08...8d3 ./test/dao/network-fee-change.ts`
- `73e...330 ./test/dao/network-fee-withdraw.ts`
- `d27...d75 ./test/dao/operational.ts`
- `994...b83 ./test/operators/others.ts`
- `689...414 ./test/operators/update-fee.ts`
- `e69...37b ./test/operators/remove.ts`
- `e7e...98c ./test/operators/register.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.9.2

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

We have executed Slither and filtered the issues that were reported and incorporated the valid ones in the report. Please note that only issues related to the scope of the audit are reported.

Test Suite Results

```
Deposit Tests
  ✓ Deposit to a non liquidated cluster I own emits "ClusterDeposited"
  ✓ Deposit to a cluster I own gas limits
  ✓ Deposit to a cluster I do not own emits "ClusterDeposited"
  ✓ Deposit to a cluster I do not own gas limits
  ✓ Deposit to a cluster I do own with a cluster that does not exist reverts "ClusterDoesNotExists"
  ✓ Deposit to a cluster I do not own with a cluster that does not exist reverts "ClusterDoesNotExists"
  ✓ Deposit to a liquidated cluster emits "ClusterDeposited"

Withdraw Tests
  ✓ Withdraw from cluster emits "ClusterWithdrawn"
  ✓ Withdraw from cluster gas limits
  ✓ Withdraw from operator balance emits "OperatorWithdrawn"
  ✓ Withdraw from operator balance gas limits
  ✓ Withdraw the total operator balance emits "OperatorWithdrawn"
  ✓ Withdraw the total operator balance gas limits
  ✓ Withdraw from a cluster that has a removed operator emits "ClusterWithdrawn"
  ✓ Withdraw more than the cluster balance reverts "InsufficientBalance"
  ✓ Withdraw from a liquidatable cluster reverts "InsufficientBalance" (liquidation threshold)
  ✓ Withdraw from a liquidatable cluster reverts "InsufficientBalance" (liquidation collateral)
  ✓ Withdraw from a liquidatable cluster after liquidation period reverts "InsufficientBalance"
  ✓ Withdraw balance from an operator I do not own reverts "CallerNotOwner"
  ✓ Withdraw more than the operator balance reverts "InsufficientBalance"
  ✓ Withdraw the total balance from an operator I do not own reverts "CallerNotOwner"
  ✓ Withdraw more than the operator total balance reverts "InsufficientBalance"
  ✓ Withdraw from a cluster without validators

Liquidation Collateral Tests
  ✓ Change minimum collateral emits "MinimumLiquidationCollateralUpdated"
  ✓ Change minimum collateral gas limits
  ✓ Get minimum collateral
  ✓ Change minimum collateral reverts "caller is not the owner"

Liquidation Threshold Tests
  ✓ Change liquidation threshold period emits "LiquidationThresholdPeriodUpdated"
  ✓ Change liquidation threshold period gas limits
  ✓ Get liquidation threshold period
  ✓ Change liquidation threshold period reverts "NewBlockPeriodIsBelowMinimum"
  ✓ Change liquidation threshold period reverts "caller is not the owner"

Network Fee Tests
  ✓ Change network fee emits "NetworkFeeUpdated"
  ✓ Change network fee when it was set emits "NetworkFeeUpdated"
  ✓ Change network fee gas limit
  ✓ Get network fee
  ✓ Change the network fee to a number below the minimum fee reverts "Max precision exceeded"
  ✓ Change the network fee to a number that exceeds allowed type limit reverts "Max value exceeded"
  ✓ Change network fee from an address thats not the DAO reverts "caller is not the owner"

DAO Network Fee Withdraw Tests
  ✓ Withdraw network earnings emits "NetworkEarningsWithdrawn"
  ✓ Withdraw network earnings gas limits
  ✓ Get withdrawable network earnings
  ✓ Get withdrawable network earnings as not owner
  ✓ Withdraw network earnings with not enough balance reverts "InsufficientBalance"
  ✓ Withdraw network earnings from an address thats not the DAO reverts "caller is not the owner"

DAO operational Tests
  ✓ Starting the transfer process does not change owner
  ✓ Ownership is transferred in a 2-step process

Deployment tests
  ✓ Upgrade SSVNetwork contract. Check new function execution
  ✓ Upgrade SSVNetwork contract. Deploy implemetation manually
  ✓ Upgrade SSVNetwork contract. Check base contract is not re-initialized
  ✓ Upgrade SSVNetwork contract. Check state is only changed from proxy contract
  ✓ Remove registerAuth from SSVNetwork contract
  ✓ Update a module (SSVOperators)

Version upgrade tests
  ✓ Upgrade contract version number

Liquidate Tests
  ✓ Liquidate a cluster via liquidation threshold emits "ClusterLiquidated"
  ✓ Liquidate a cluster via minimum liquidation collateral emits "ClusterLiquidated"
  ✓ Liquidate a cluster after liquidation period emits "ClusterLiquidated"
  ✓ Liquidatable with removed operator
  ✓ Liquidatable with removed operator after liquidation period
  ✓ Liquidate validator with removed operator in a cluster
  ✓ Liquidate and register validator in a disabled cluster reverts "ClusterIsLiquidated"
  ✓ Liquidate cluster (4 operators) and check isLiquidated true
  ✓ Liquidate cluster (7 operators) and check isLiquidated true
  ✓ Liquidate cluster (10 operators) and check isLiquidated true
  ✓ Liquidate cluster (13 operators) and check isLiquidated true
  ✓ Liquidate a non liquidatable cluster that I own
  ✓ Liquidate cluster that I own
  ✓ Liquidate cluster that I own after liquidation period
  ✓ Get if the cluster is liquidatable
  ✓ Get if the cluster is liquidatable after liquidation period
  ✓ Get if the cluster is not liquidatable
  ✓ Liquidate a cluster that is not liquidatable reverts "ClusterNotLiquidatable"
  ✓ Liquidate a cluster that is not liquidatable reverts "IncorrectClusterState"
  ✓ Liquidate already liquidated cluster reverts "ClusterIsLiquidated"
```

```
    ✓ Is liquidated reverts "ClusterDoesNotExists"

  Liquidate Tests
    ✓ Liquidate -> deposit -> reactivate
    ✓ RegisterValidator -> liquidate -> removeValidator -> deposit -> withdraw
    ✓ Withdraw -> liquidate -> deposit -> reactivate

  Reactivate Tests
    ✓ Reactivate a disabled cluster emits "ClusterReactivated"
    ✓ Reactivate a cluster with a removed operator in the cluster
    ✓ Reactivate an enabled cluster reverts "ClusterAlreadyEnabled"
    ✓ Reactivate a cluster when the amount is not enough reverts "InsufficientBalance"
    ✓ Reactivate a liquidated cluster after making a deposit
    ✓ Reactivate a cluster after liquidation period when the amount is not enough reverts
"InsufficientBalance"

  Others Operator Tests
    ✓ Add fee recipient address emits "FeeRecipientAddressUpdated"
    ✓ Remove operator whitelisted address
    ✓ Non-owner remove operator whitelisted address reverts "CallerNotOwner"
    ✓ Update operator whitelisted address
    ✓ Non-owner update operator whitelisted address reverts "CallerNotOwner"
    ✓ Get the maximum number of validators per operator

  Register Auth Operator Tests
    ✓ Register auth and get auth data
    ✓ Register operator with unauthorized address reverts "NotAuthorized"
    ✓ Register operator with unauthorized address reverts "NotAuthorized" (2)
    ✓ Register validator with unauthorized address reverts "NotAuthorized"
    ✓ Register validator with unauthorized address reverts "NotAuthorized" (2)

  Register Operator Tests
    ✓ Register operator emits "OperatorAdded"
    ✓ Register operator gas limits
    ✓ Get operator by id
    ✓ Get private operator by id
    ✓ Set operator whitelist gas limits
    ✓ Get non-existent operator by id
    ✓ Get operator removed by id
    ✓ Register an operator with a fee thats too low reverts "FeeTooLow"
    ✓ Register same operator twice reverts "OperatorAlreadyExists"

  Remove Operator Tests
    ✓ Remove operator emits "OperatorRemoved"
    ✓ Remove private operator emits "OperatorRemoved"
    ✓ Remove operator gas limits
    ✓ Remove operator with 0 balance emits "OperatorWithdrawn"
    ✓ Remove operator with a balance emits "OperatorWithdrawn"
    ✓ Remove operator with a balance gas limits
    ✓ Remove operator I do not own reverts "CallerNotOwner"
    ✓ Remove same operator twice reverts "OperatorDoesNotExist"

  Operator Fee Tests
    ✓ Declare fee emits "OperatorFeeDeclared"
    ✓ Declare fee gas limits"
    ✓ Declare fee with zero value emits "OperatorFeeDeclared"
    ✓ Declare a lower fee gas limits
    ✓ Declare a higher fee gas limit
    ✓ Cancel declared fee emits "OperatorFeeDeclarationCancelled"
    ✓ Cancel declared fee gas limits
    ✓ Execute declared fee emits "OperatorFeeExecuted"
    ✓ Execute declared fee gas limits
    ✓ Get operator fee
    ✓ Get fee from operator that does not exist returns 0
    ✓ Declare fee of operator I do not own reverts "CallerNotOwner"
    ✓ Declare fee with a wrong Publickey reverts "OperatorDoesNotExist"
    ✓ Declare fee when previously set to zero reverts "FeeIncreaseNotAllowed"
    ✓ Declare same fee value as actual reverts "SameFeeChangeNotAllowed"
    ✓ Declare fee after registering an operator with zero fee reverts "FeeIncreaseNotAllowed"
    ✓ Declare fee above the operators max fee increase limit reverts "FeeExceedsIncreaseLimit"
    ✓ Cancel declared fee without a pending request reverts "NoFeeDeclared"
    ✓ Cancel declared fee of an operator I do not own reverts "CallerNotOwner"
    ✓ Execute declared fee of an operator I do not own reverts "CallerNotOwner"
    ✓ Execute declared fee without a pending request reverts "NoFeeDeclared"
    ✓ Execute declared fee too early reverts "ApprovalNotWithinTimeframe"
    ✓ Execute declared fee too late reverts "ApprovalNotWithinTimeframe"
    ✓ Reduce fee emits "OperatorFeeExecuted"
    ✓ Reduce fee emits "OperatorFeeExecuted"
    ✓ Reduce fee with an increased value reverts "FeeIncreaseNotAllowed"
    ✓ Reduce fee after declaring a fee change
    ✓ DAO increase the fee emits "OperatorFeeIncreaseLimitUpdated"
    ✓ DAO increase the fee gas limits"
    ✓ DAO update the declare fee period emits "DeclareOperatorFeePeriodUpdated"
    ✓ DAO update the declare fee period gas limits"
    ✓ DAO update the execute fee period emits "ExecuteOperatorFeePeriodUpdated"
    ✓ DAO update the execute fee period gas limits
    ✓ DAO get fee increase limit
    ✓ DAO get declared fee
    ✓ DAO get declared and execute fee periods
    ✓ Increase fee from an address thats not the DAO reverts "caller is not the owner"
    ✓ Update the declare fee period from an address thats not the DAO reverts "caller is not the owner"
    ✓ Update the execute fee period from an address thats not the DAO reverts "caller is not the owner"
    ✓ DAO declared fee without a pending request reverts "NoFeeDeclared"

  Balance Tests
    ✓ Check cluster balance in three blocks, one after the other
    ✓ Check cluster balance in two and twelve blocks, after network fee updates
    ✓ Check DAO earnings in three blocks, one after the other
    ✓ Check DAO earnings in two and twelve blocks, after network fee updates
    ✓ Check operators earnings in three blocks, one after the other
    ✓ Check cluster balance with removed operator
    ✓ Check cluster balance with not enough balance
    ✓ Check cluster balance in a non liquidated cluster
    ✓ Check cluster balance in a liquidated cluster reverts "ClusterIsLiquidated"
    ✓ Check operator earnings, cluster balances and network earnings"
    ✓ Check operator earnings and cluster balance when reducing operator fee"
    ✓ Check cluster balance after withdraw and deposit"

  Register Validator Tests
    ✓ Register validator with 4 operators emits "ValidatorAdded"
    ✓ Register validator with 4 operators gas limit
    ✓ Register 2 validators into the same cluster gas limit
    ✓ Register 2 validators into the same cluster and 1 validator into a new cluster gas limit
    ✓ Register 2 validators into the same cluster with one time deposit gas limit
    ✓ Register validator with 7 operators gas limit
    ✓ Register 2 validators with 7 operators into the same cluster gas limit
    ✓ Register 2 validators with 7 operators into the same cluster and 1 validator into a new cluster
with 7 operators gas limit
    ✓ Register 2 validators with 7 operators into the same cluster with one time deposit gas limit
    ✓ Register validator with 10 operators gas limit
    ✓ Register 2 validators with 10 operators into the same cluster gas limit
    ✓ Register 2 validators with 10 operators into the same cluster and 1 validator into a new cluster
with 10 operators gas limit
    ✓ Register 2 validators with 10 operators into the same cluster with one time deposit gas limit
    ✓ Register validator with 13 operators gas limit
    ✓ Register 2 validators with 13 operators into the same cluster gas limit
    ✓ Register 2 validators with 13 operators into the same cluster and 1 validator into a new cluster
with 13 operators gas limit
    ✓ Register 2 validators with 13 operators into the same cluster with one time deposit gas limit
    ✓ Get cluster burn rate
    ✓ Get cluster burn rate when one of the operators does not exist
    ✓ Register validator with incorrect input data reverts "IncorrectClusterState"
    ✓ Register validator in a new cluster with incorrect input data reverts "IncorrectClusterState"
    ✓ Register validator when an operator does not exist in the cluster reverts "OperatorDoesNotExist"
    ✓ Register validator with a removed operator in the cluster reverts "OperatorDoesNotExist"
    ✓ Register cluster with unsorted operators reverts "UnsortedOperatorsList"
    ✓ Register cluster with duplicated operators reverts "OperatorsListNotUnique"
    ✓ Register validator into a cluster with an invalid amount of operators reverts
"InvalidOperatorIdsLength"
    ✓ Register validator with an invalid public key length reverts "InvalidPublicKeyLength"
    ✓ Register validator with not enough balance reverts "InsufficientBalance"
    ✓ Register validator in a liquidatable cluster with not enough balance reverts "InsufficientBalance"
    ✓ Register an existing validator with same operators setup reverts "ValidatorAlreadyExists"
    ✓ Register an existing validator with different operators setup reverts "ValidatorAlreadyExists"
    ✓ Surpassing max number of validators per operator reverts "ExceedValidatorLimit"
    ✓ Register whitelisted validator in 1 operator with 4 operators emits "ValidatorAdded"
    ✓ Register a non whitelisted validator reverts "CallerNotWhitelisted"
    ✓ Retrieve an existing validator
    ✓ Retrieve a non-existing validator

  Remove Validator Tests
    ✓ Remove validator emits "ValidatorRemoved"
    ✓ Remove validator after cluster liquidation period emits "ValidatorRemoved"
    ✓ Remove validator gas limit
    ✓ Remove validator with a removed operator in the cluster
    ✓ Register a removed validator and remove the same validator again
    ✓ Remove validator from a liquidated cluster
    ✓ Remove validator with an invalid owner reverts "ValidatorDoesNotExist"
    ✓ Remove validator with an invalid operator setup reverts "IncorrectValidatorState"
    ✓ Remove the same validator twice reverts "ValidatorDoesNotExist"

 209 passing (3m)
```

Code Coverage

The test suite has high branch coverage, but we recommend adding additional tests to confirm for each issue resolution during fix review.

Fix Review: Following the codebase's architecture overhaul, the code coverage has reduced. We recommend improving the test suite. The code coverage was obtained using `npm run solidity-coverage`.

Fix Review 2: The client added additional tests. The code coverage did not change significantly. We recommend improving the test suite.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	75	98.11	100	
SSVNetwork.sol	100	83.33	100	100	
SSVNetworkViews.sol	100	33.33	95	100	
SSVProxy.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
ISSVClusters.sol	100	100	100	100	
ISSVDAO.sol	100	100	100	100	
ISSVNetwork.sol	100	100	100	100	
ISSVNetworkCore.sol	100	100	100	100	
ISSVOperators.sol	100	100	100	100	
ISSVViews.sol	100	100	100	100	
contracts/libraries/	97.96	83.33	100	93.42	
ClusterLib.sol	100	100	100	100	
CoreLib.sol	88.89	50	100	76.92	15,21,44
OperatorLib.sol	100	90	100	95.45	49
ProtocolLib.sol	100	75	100	91.67	43

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
RegisterAuth.sol	100	100	100	100	
SSVStorage.sol	100	100	100	100	
SSVStorageProtocol.sol	100	100	100	100	
Types.sol	100	100	100	100	
contracts/mocks/	100	50	100	100	
SSVTokenMock.sol	100	50	100	100	
contracts/modules/	99.55	90.52	97.56	99.68	
SSVClusters.sol	100	86.76	100	100	
SSVDAO.sol	100	100	100	100	
SSVOperators.sol	100	97.06	100	100	
SSVViews.sol	98.11	90	94.12	98.25	201
contracts/test/	6.82	7.14	10	9.09	
SSVNetworkBasicUpgrade.sol	100	100	100	100	
SSVNetworkReinitializable.sol	100	0	0	0	9
SSVNetworkUpgrade.sol	17.65	7.69	10.34	17.02	... 325,332,353
SSVNetworkValidatorsPerOperator.sol	100	50	100	100	
SSVViewsT.sol	0	0	0	0	... 197,201,205
contracts/test/interfaces/	100	100	100	100	
ISSVNetworkT.sol	100	100	100	100	
contracts/test/libraries/	0	0	0	0	
CoreLibT.sol	0	0	0	0	9,13,14
SSVStorageT.sol	0	100	0	0	39,40
contracts/test/modules/	1.92	3.57	9.09	1.3	
SSVOperatorsUpdate.sol	1.92	3.57	9.09	1.3	... 175,179,180
All files	70.76	63.74	67.74	71.23	

Changelog

- 2023-03-24 - Initial report
- 2023-07-07 - Fix review
- 2023-07-21 - Fix review 2

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Quantstamp

SSV.network

Quantstamp