Quantstamp

SSV Network - Diff Audit

Executive Summary

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

Type	Registry and Payment
Timeline	2023-10-23 through 2023-10-30
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	SIP: Voluntary Exit
Source Code	• bloxapp/ssv-network #ca3ad7f
Auditors	• Jennifer Wu Auditing Engineer • Roman Rohleder Senior Auditing Engineer • Cameron Biniamow Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	10 Fixed: 6 Acknowledged: 4
High severity findings	1 Fixed: 1
Medium severity findings	0
Low severity findings	4 Acknowledged: 4
Undetermined severity findings	0
Informational findings	5 Fixed: 5

Summary of Findings

This audit report is a **diff** audit, highlighting the changes made between **v1.0.0-rc3** and **v1.0.0**. It is crucial for readers to review this diff audit alongside the final audit report for SSV.network, as the scope of this diff audit is strictly limited to changes between v1.0.0-rc3 and v1.0.0. Between v1.0.0-rc3 and v1.0.0, the contracts underwent minor bug fixes related to network earnings withdrawals and the `Types.shrink()` function and introduced a new feature for a validator to voluntarily exit through the function `exitValidator()`. This function emits the event `ValidatorExited` and initiates an off-chain process to exit from SSV.network, which is out of scope as the diff audit is limited to smart contracts only.

The diff audit resulted in 10 findings: 1 high and 4 lows and 5 gas optimization suggestions outlined below. The gas optimization issues are suggestions to improve gas efficiency without major refactoring. We recommend the client to consider all identified issues.

Fix Review: During the fix review, the client fixed SSV-1 by adding `msg.sender` when emitting the event `ValidatorExited`; the `msg.sender` will be validated by the off-chain process before the validator exit process is initiated. The client resolved remaining issues SSV-2 to SSV-10 by implementing fixes or acknowledging them.

ID	DESCRIPTION	SEVERITY	STATUS
SSV-1	Force Validators to Exit	High	Fixed
SSV-2	Colluding Operators Can Act Maliciously on Behalf of a Validator	Low	Acknowledged
SSV-3	Front Run Operator Registration	Low	Acknowledged
SSV-4	Event `ValidatorExited` Can Be Emitted Multiple Times	Low	Acknowledged
SSV-5	Missing Input Validation	Low	Acknowledged
SSV-6	Gas Optimization: Wasted Deployment Gas From Unused Named Return Variables	Informational	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
SSV-8	Gas Optimization: Use `calldata` Instead of `memory`	Informational	Fixed
SSV-9	Gas Optimization: Inefficient Storage Clearing Patterns	Informational	Fixed
SSV-10	Gas Optimization: Cache Variables	Informational	Fixed

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 changes between v1.0.0-rc3 and v1.0.0 for the smart contracts supporting 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.

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code 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 Force Validators to Exit

High Fixed

> **Update**
> The client fixed the issue in `e696b3a441d93956651da4d404ef713f543f7158` by adding `msg.sender` in the event `ValidatorExited` to be validated off-chain.

File(s) affected: `SSVClusters.sol`

Description: The `ValidatorExited` event, emitted by the `exitValidator()` function, initiates a validator's exit from the network. The off-chain component parses the data from the event and starts the exit process. However, since the event only emits the public key and the operators, anyone with a copy of the public key can initiate the process. Validators can be registered multiple times using the same public key with different `msg.sender` addresses, creating separate entries for what should be a single validator. Although the registered validator is invalid off-chain, it is valid on-chain. This situation becomes problematic when a malicious actor invokes `exitValidator()` using the public key of a legitimate validator and a different `msg.sender`, as the function does not verify ownership of the associated private key, potentially initiating an unauthorized exit.

Exploit Scenario:

1. Alice registers as a validator using `registerValidator()`.
2. Eve copies Alice's transaction and registers as a validator. The validator hash is different due to different `msg.sender` so the registration is valid on-chain but invalid off-chain due to invalid signed shares data.
3. Eve forces Alice to voluntarily exit by calling `exitValidator()`.
4. The off-chain component picks up the event `ValidatorExited(publicKey, operatorIds)` and initiates the exit process.

Recommendation: The event `ValidatorExited` should emit `msg.sender` to be validated by the off-chain process before initiating the exit process.

SSV-2 Colluding Operators Can Act Maliciously on Behalf of a Validator

Low Acknowledged

> **Update**
> The client acknowledged the issue and provided the following explanation:
>
> > It is critical that validators choose trusted and decentralized operators when registering.

File(s) affected: `SSVNetwork.sol`

Description: From the final audit report for the SSV.network (report-SSV-17), the SSV team addressed this issue by manually authorizing operators. However, the current implementation has eliminated the need for prior authorization, reopening the possibility for colluding operators within the same cluster to maliciously act on behalf of the validator.

Recommendation: It is critical that validators choose trusted and decentralized operators when registering.

SSV-3 Front Run Operator Registration

Low Acknowledged

> **Update**
> The client acknowledged the issue and provided the following explanation:
>
> > 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`

Description: From the final audit report for the SSV.network (report-SSV-18), with the removal of authorization to register operators, the operator registration is now susceptible to front-running. A malicious actor could exploit this `SSVNetwork.registerOperator()` vulnerability by copying and front-running the legitimate operator registration transaction and consequently blocking the legitimate operator from registration.

Recommendation: Consider including `msg.sender` in the public key hashing, as is done for validators.

SSV-4 Event `ValidatorExited` Can Be Emitted Multiple Times

Low | Acknowledged

> **Update**
> The client explained that the event `ValidatorExited` can be emitted multiple times and corrected the voluntary_exit.md specification.

File(s) affected: `SSVClusters.sol`

Description: The `exitValidator()` function allows for the `ValidatorExited` event to be emitted multiple times for the same validator, as there is no on-chain check to ensure it is called only once. This goes against the specification which states that the `exitValidator()` can be called only once by the validator owner.

Recommendation:

1. Implement an on-chain mechanism using a mapping or state variable to track validators that have already exited. Prior to emitting the `ValidatorExited` event, consult this mechanism to ensure that the `exitValidator()` function has not been previously called for the same validator.
2. Alternatively, if the "only once" limitation is enforced off-chain, we recommend clarifying this enforcement in the contract's code documentation.

SSV-5 Missing Input Validation

Low | Acknowledged

> **Update**
> The client acknowledged the issue and provided the following explanation:
>
> As this does not represent a security risk, we delegate the responsibility on the caller to provide the right parameters.

File(s) affected: `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.

Following is the list of places that can potentially benefit from stricter input validation:

1. `SSVDAO.sol#25` : the `amount` of the `withdrawNetworkEarnings()` should be greater than zero.

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

SSV-6 Gas Optimization: Wasted Deployment Gas From Unused Named Return Variables

Informational | Fixed

> **Update**
> The client fixed the issue as per recommendation in `c1ba481bf57267b8504e1a2cb6ad7d18fdc120c4` .

File(s) affected: `SSVViews.sol`

Description: Functions that declare named return variables but return values separately consume more gas during deployment. Consider modifying the following functions to remove unused named return variables:

1. Fixed `SSVViews.getValidator()`
2. Fixed `SSVViews.getOperatorFee()`
3. Fixed `SSVViews.getOperatorFeeIncreaseLimit()`
4. Fixed `SSVViews.getMaximumOperatorFee()`
5. Fixed `SSVViews.getOperatorFeePeriods()`
6. Fixed `SSVViews.getVersion()`

Recommendation: Remove the unused named return variables for gas optimization.

SSV-7 Gas Optimization: Use Constant Instead of `type(uint).max`

Informational | Fixed

> **Update**
> The client reviewed the issue and provided the following explanation:

The Ethereum Virtual Machine (EVM) does not differentiate between a value that is defined in the contract (or library) as a constant and a value that comes from Solidity's type system; both are immutable and known at the time of compilation, thus they are inlined in the compiled bytecode. As a result, there's no extra computational overhead during execution that would cause a difference in gas usage.

We agree with the client's explanation and confirm that this issue is a false positive gas optimization recommendation. We also corrected the issue description. Upon a detailed examination of the bytecode, we observed a slight difference in the generated opcodes when using `type(uint32).max` as opposed to a constant value. Specifically, when optimization is not enabled, using `type(uint32).max` introduces two additional opcodes `DUP1` and `AND`, resulting in an extra gas cost of 6 units during execution. It is important to note that this difference gets optimized out and becomes irrelevant when the compiler optimization is enabled. Since the SSV protocol uses optimization during compilation, this issue is not relevant.

File(s) affected: `ProtocolLib.sol`

Description: The use of `type(uintx).max` in Solidity generates bytecode that includes additional opcodes compared to using a constant value directly. Specifically, when using `type(uintx).max`, the generated bytecode includes additional `DUP1` and `AND` opcodes, resulting in a minor increase in gas cost (6 gas units) for each occurrence. It is important to note that this difference becomes negligible when compiler optimization is enabled, as the additional opcodes are optimized away.

While the Ethereum Virtual Machine (EVM) treats compile-time constants and values derived from Solidity's type system in a similar manner (both are resolved during compilation), subtle differences in the generated bytecode can lead to minor discrepancies in gas usage.

Recommendation: For functions where every unit of gas is critical, and to ensure consistency in gas usage regardless of compiler optimization settings, you can define a constant for the maximum value of `uint32` and use it in place of `type(uint32).max`. This ensures that the generated bytecode is as efficient as possible, even when compiler optimization is not enabled.

SSV-8 Gas Optimization: Use `calldata` Instead of `memory`

Informational | Fixed

Update
The client fixed the issue as per recommendation in `c70a5138b6916b3e5ad3bd4599db3d37cf9d573b`.

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

Description: Solidity's `calldata` is a read-only byte-addressable space where function arguments reside. It is exclusive to external function call parameters, and it is more cost-effective to employ `calldata` over `memory`. This is because `calldata` is not stored in memory but is directly accessed from the function call data, resulting in gas savings. It is recommended to change the `memory` to `calldata` in the mentioned functions:

1. Fixed `operatorIds` parameter in `SSVClusters.liquidate()`
2. Fixed `operatorIds` parameter in `ISSVViews.isLiquidatable()`

Recommendation: Change from `memory` to `calldata`.

SSV-9 Gas Optimization: Inefficient Storage Clearing Patterns

Informational | Fixed

Update
The client fixed the issue as per recommendation in `f7e59b96ae4d1d4e4083122b26ec78a75286bd09`.

File(s) affected: `SSVOperators.sol`

Description: Some parts of the code in `SSVOperators.sol` can be optimized for gas usage by simplifying storage clearing patterns:

1. Fixed When clearing the whitelist for an `operatorId` in the function `SSVOperators.removeOperator()`, instead of checking whether the `s.operatorsWhitelist` address is initialized, delete `s.operatorsWhitelist` directly.
2. Fixed When clearing the `approvalBeginTime` in the function `SSVOperators.reduceOperatorFee()`, instead of checking whether the `approvalBeginTime` is initialized, delete `approvalBeginTime` directly.

Recommendation: Clearing storage in Ethereum provides a gas refund, promoting efficient storage utilization. By omitting unnecessary checks before deletion, gas consumption can be reduced, offering more efficient transactions. Consider updating the functions as described to improve gas optimization.

SSV-10 Gas Optimization: Cache Variables

Informational | Fixed

Update
The client fixed the issue as per recommendation in `673191ca91ce0b941f06196c1b9a26b3bc36347f`.

File(s) affected: `OperatorLib.sol`, `SSVDAO.sol`

Description: Repeatedly accessing certain variables can be gas-intensive. To optimize gas usage, values frequently accessed should be stored in memory variables. Consider storing the value in a memory variable and reference the memory variable for the following variables:

1. Fixed The usage of `operatorIds.length` in `OperatorLib.updateOperators()` can be stored in a local variable.
2. Fixed The usage of `SSVStorageProtocol.load().validatorsPerOperatorLimit` in `OperatorLib.updateOperators()` can be refactored to leverage the `StorageProtocol` storage pointer and pass as a function parameter.
3. Fixed Instead of expanding the shrunken `fee` in the function `SSVDAO.updateNetworkFee()` the event `NetworkFeeUpdated` can emit the `fee` because the `fee` can be only shrunken if the `fee` passes `Types256.shrinkable()`.

Recommendation: Consider applying variable caching as per the recommendation provided in the issue.

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).

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

- `59c...d99 ./contracts/SSVNetwork.sol`
- `b2a...858 ./contracts/SSVNetworkViews.sol`
- `249...f2e ./contracts/SSVProxy.sol`
- `001...894 ./contracts/interfaces/ISSVOperators.sol`
- `906...40e ./contracts/interfaces/ISSVNetwork.sol`
- `99c...1d4 ./contracts/interfaces/ISSVViews.sol`
- `014...6dc ./contracts/interfaces/ISSVDAO.sol`
- `09a...c1d ./contracts/interfaces/ISSVClusters.sol`
- `df8...d65 ./contracts/interfaces/ISSVNetworkCore.sol`
- `1a8...cd8 ./contracts/modules/SSVOperators.sol`
- `498...d0b ./contracts/modules/SSVViews.sol`
- `0ce...bd4 ./contracts/modules/SSVClusters.sol`
- `3d6...c50 ./contracts/modules/SSVDAO.sol`
- `237...e14 ./contracts/libraries/SSVStorage.sol`
- `7ea...f65 ./contracts/libraries/SSVStorageProtocol.sol`
- `e6e...607 ./contracts/libraries/ValidatorLib.sol`
- `974...702 ./contracts/libraries/OperatorLib.sol`
- `91b...26a ./contracts/libraries/CoreLib.sol`
- `a3b...9d6 ./contracts/libraries/ClusterLib.sol`
- `cef...bc5 ./contracts/libraries/ProtocolLib.sol`

- `2ad...ade ./contracts/libraries/Types.sol`

Tests

- `1eb...316 ./test/helpers/utils.ts`
- `22c...649 ./test/helpers/gas-usage.ts`
- `10c...6f3 ./test/helpers/contract-helpers.ts`
- `4ea...9e3 ./test/account/deposit.ts`
- `856...01b ./test/account/withdraw.ts`
- `885...3f5 ./test/validators/others.ts`
- `ec6...ca8 ./test/validators/remove.ts`
- `34e...b7d ./test/validators/register.ts`
- `cb6...b4f ./test/deployment/version.ts`
- `97d...765 ./test/deployment/deploy.ts`
- `49b...f41 ./test/sanity/balances.ts`
- `bb3...19e ./test/liquidate/liquidate.ts`
- `889...91e ./test/liquidate/liquidated-cluster.ts`
- `b16...5db ./test/liquidate/reactivate.ts`
- `273...e7f ./test/dao/liquidation-collateral.ts`
- `314...9c7 ./test/dao/liquidation-threshold.ts`
- `ceb...0f7 ./test/dao/network-fee-change.ts`
- `995...092 ./test/dao/network-fee-withdraw.ts`
- `f85...c34 ./test/dao/operational.ts`
- `fb4...01a ./test/operators/others.ts`
- `534...8c8 ./test/operators/update-fee.ts`
- `28c...4e1 ./test/operators/remove.ts`
- `7cb...b67 ./test/operators/register.ts`

Automated Analysis

N/A

Test Suite Results

```
  Deposit Tests
    ✔ Deposit to a non liquidated cluster I own emits "ClusterDeposited" (62ms)
    ✔ Deposit to a cluster I own gas limits (50ms)
    ✔ Deposit to a cluster I do not own emits "ClusterDeposited" (47ms)
    ✔ Deposit to a cluster I do not own gas limits (55ms)
    ✔ Deposit to a cluster I do own with a cluster that does not exist reverts "ClusterDoesNotExists"
(41ms)
    ✔ Deposit to a cluster I do not own with a cluster that does not exist reverts "ClusterDoesNotExists"
    ✔ Deposit to a liquidated cluster emits "ClusterDeposited" (108ms)

  Withdraw Tests
    ✔ Withdraw from cluster emits "ClusterWithdrawn" (38ms)
    ✔ Withdraw from cluster gas limits (38ms)
    ✔ 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" (62ms)
    ✔ Withdraw more than the cluster balance reverts "InsufficientBalance" (54ms)
    ✔ Sequentially withdraw more than the cluster balance reverts "InsufficientBalance" (224ms)
    ✔ Withdraw from a liquidatable cluster reverts "InsufficientBalance" (liquidation threshold) (46ms)
    ✔ Withdraw from a liquidatable cluster reverts "InsufficientBalance" (liquidation collateral) (42ms)
    ✔ Withdraw from a liquidatable cluster after liquidation period reverts "InsufficientBalance" (45ms)
    ✔ Withdraw balance from an operator I do not own reverts "CallerNotOwner"
    ✔ Withdraw more than the operator balance reverts "InsufficientBalance"
    ✔ Sequentially withdraw more than the operator balance reverts "InsufficientBalance" (88ms)
```

```
  ✔ 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 (83ms)

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 providing UINT64 max value reverts "Max value exceeded"
  ✔ Change network fee when it was set emits "NetworkFeeUpdated" (44ms)
  ✔ 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"
  ✔ Withdraw network earnings providing UINT64 max value reverts "Max value exceeded"
  ✔ Withdraw network earnings sequentially when not enough balance reverts "InsufficientBalance" (81ms)

DAO operational Tests
  ✔ Starting the transfer process does not change owner
  ✔ Ownership is transferred in a 2-step process
  ✔ Get the network validators count (add/remove validaotor) (223ms)
  ✔ Get the network validators count (add/remove validaotor) (227ms)

Deployment tests
  ✔ Check default values after deploying
  ✔ Upgrade SSVNetwork contract. Check new function execution (115ms)
  ✔ Upgrade SSVNetwork contract. Deploy implemetation manually (63ms)
  ✔ Upgrade SSVNetwork contract. Check base contract is not re-initialized (48ms)
  ✔ Upgrade SSVNetwork contract. Check state is only changed from proxy contract (53ms)
  ✔ Update a module (SSVOperators) (65ms)
  ✔ ETH can not be transferred to SSVNetwork / SSVNetwork views

Version upgrade tests
  ✔ Upgrade contract version number (58ms)

Liquidate Tests
  ✔ Liquidate a cluster via liquidation threshold emits "ClusterLiquidated" (50ms)
  ✔ Liquidate a cluster via minimum liquidation collateral emits "ClusterLiquidated" (48ms)
  ✔ Liquidate a cluster after liquidation period emits "ClusterLiquidated" (43ms)
  ✔ Liquidatable with removed operator (46ms)
  ✔ Liquidatable with removed operator after liquidation period (45ms)
  ✔ Liquidate validator with removed operator in a cluster (79ms)
  ✔ Liquidate and register validator in a disabled cluster reverts "ClusterIsLiquidated" (99ms)
  ✔ Liquidate cluster (4 operators) and check isLiquidated true (58ms)
  ✔ Liquidate cluster (7 operators) and check isLiquidated true (163ms)
  ✔ Liquidate cluster (10 operators) and check isLiquidated true (193ms)
  ✔ Liquidate cluster (13 operators) and check isLiquidated true (223ms)
  ✔ Liquidate a non liquidatable cluster that I own (55ms)
  ✔ Liquidate cluster that I own (61ms)
  ✔ Liquidate cluster that I own after liquidation period (56ms)
  ✔ 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" (66ms)
    ✔ Liquidate a cluster that is not liquidatable reverts "IncorrectClusterState"
    ✔ Liquidate already liquidated cluster reverts "ClusterIsLiquidated" (71ms)
    ✔ Is liquidated reverts "ClusterDoesNotExists" (60ms)

  Liquidate Tests
    ✔ Liquidate -> deposit -> reactivate (178ms)
    ✔ RegisterValidator -> liquidate -> removeValidator -> deposit -> withdraw (218ms)
    ✔ Withdraw -> liquidate -> deposit -> reactivate (288ms)
    ✔ Remove validator -> withdraw -> try liquidate reverts "ClusterNotLiquidatable" (215ms)

  Reactivate Tests
    ✔ Reactivate a disabled cluster emits "ClusterReactivated" (110ms)
    ✔ Reactivate a cluster with a removed operator in the cluster (130ms)
    ✔ Reactivate an enabled cluster reverts "ClusterAlreadyEnabled"
    ✔ Reactivate a cluster when the amount is not enough reverts "InsufficientBalance" (91ms)
    ✔ Reactivate a liquidated cluster after making a deposit (141ms)
    ✔ Reactivate a cluster after liquidation period when the amount is not enough reverts
"InsufficientBalance" (105ms)

  Others Operator Tests
    ✔ Add fee recipient address emits "FeeRecipientAddressUpdated"
    ✔ Remove operator whitelisted address (63ms)
    ✔ Non-owner remove operator whitelisted address reverts "CallerNotOwner" (60ms)
    ✔ Update operator whitelisted address (44ms)
    ✔ Non-owner update operator whitelisted address reverts "CallerNotOwner" (39ms)
    ✔ Get the maximum number of validators per operator

  Register Operator Tests
    ✔ Register operator emits "OperatorAdded"
    ✔ Register operator gas limits
    ✔ Get operator by id
    ✔ Get private operator by id (55ms)
    ✔ Set operator whitelist gas limits (44ms)
    ✔ Get non-existent operator by id
    ✔ Get operator removed by id (57ms)
    ✔ Register an operator with a fee thats too low reverts "FeeTooLow"
    ✔ Register an operator with a fee thats too high reverts "FeeTooHigh"
    ✔ Register same operator twice reverts "OperatorAlreadyExists" (43ms)

  Remove Operator Tests
    ✔ Remove operator emits "OperatorRemoved"
    ✔ Remove private operator emits "OperatorRemoved" (71ms)
    ✔ Remove operator gas limits
    ✔ Remove operator with 0 balance emits "OperatorWithdrawn"
    ✔ Remove operator with a balance emits "OperatorWithdrawn" (97ms)
    ✔ Remove operator with a balance gas limits (100ms)
    ✔ Remove operator I do not own reverts "CallerNotOwner"
    ✔ Remove same operator twice reverts "OperatorDoesNotExist" (42ms)

  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" (42ms)
    ✔ Cancel declared fee gas limits (43ms)
    ✔ Execute declared fee emits "OperatorFeeExecuted" (47ms)
    ✔ Execute declared fee gas limits (48ms)
    ✔ Get operator fee
    ✔ Get fee from operator that does not exist returns 0
    ✔ Get operator maximum fee limit
    ✔ 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" (66ms)
    ✔ Declare same fee value as actual reverts "SameFeeChangeNotAllowed" (64ms)
    ✔ Declare fee after registering an operator with zero fee reverts "FeeIncreaseNotAllowed" (41ms)
    ✔ Declare fee above the operators max fee increase limit reverts "FeeExceedsIncreaseLimit"
    ✔ Declare fee above the operators max fee limit reverts "FeeTooHigh"
    ✔ Declare fee too high reverts "FeeTooHigh" -> DAO updates limit -> declare fee emits
```

```
"OperatorFeeDeclared" (103ms)
    ✔ Cancel declared fee without a pending request reverts "NoFeeDeclared"
    ✔ Cancel declared fee of an operator I do not own reverts "CallerNotOwner" (39ms)
    ✔ Execute declared fee of an operator I do not own reverts "CallerNotOwner" (47ms)
    ✔ Execute declared fee without a pending request reverts "NoFeeDeclared"
    ✔ Execute declared fee too early reverts "ApprovalNotWithinTimeframe" (44ms)
    ✔ Execute declared fee too late reverts "ApprovalNotWithinTimeframe" (41ms)
    ✔ Reduce fee emits "OperatorFeeExecuted" (57ms)
    ✔ Reduce fee emits "OperatorFeeExecuted"
    ✔ Reduce fee with an increased value reverts "FeeIncreaseNotAllowed"
    ✔ Reduce fee after declaring a fee change (54ms)
    ✔ Reduce maximum fee limit after declaring a fee change reverts "FeeTooHigh (60ms)
    ✔ DAO increase the fee emits "OperatorFeeIncreaseLimitUpdated"
    ✔ DAO update the maximum operator fee emits "OperatorMaximumFeeUpdated"
    ✔ 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 update the maximum fee for operators using SSV 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 (62ms)
    ✔ Check cluster balance in two and twelve blocks, after network fee updates (108ms)
    ✔ Check DAO earnings in three blocks, one after the other
    ✔ Check DAO earnings in two and twelve blocks, after network fee updates (47ms)
    ✔ Check operators earnings in three blocks, one after the other (76ms)
    ✔ Check cluster balance with removed operator (48ms)
    ✔ Check cluster balance with not enough balance
    ✔ Check cluster balance in a non liquidated cluster
    ✔ Check cluster balance in a liquidated cluster reverts "ClusterIsLiquidated" (69ms)
    ✔ Check operator earnings, cluster balances and network earnings" (336ms)
    ✔ Check operator earnings and cluster balance when reducing operator fee" (55ms)
    ✔ Check cluster balance after withdraw and deposit" (252ms)

  Other Validator Tests
    ✔ Exiting a validator emits "ValidatorExited"
    ✔ Exiting a validator gas limit
    ✔ Exiting one of the validators in a cluster emits "ValidatorExited" (100ms)
    ✔ Exiting a removed validator reverts "ValidatorDoesNotExist" (69ms)
    ✔ Exiting a non-existing validator reverts "ValidatorDoesNotExist"
    ✔ Exiting a validator with empty operator list reverts "IncorrectValidatorState"
    ✔ Exiting a validator with empty public key reverts "ValidatorDoesNotExist"
    ✔ Exiting a validator using the wrong account reverts "ValidatorDoesNotExist"
    ✔ Exiting a validator with incorrect operators (unsorted list) reverts with "IncorrectValidatorState"
    ✔ Exiting a validator with incorrect operators (too many operators) reverts with
"IncorrectValidatorState" (155ms)
    ✔ Exiting a validator with incorrect operators reverts with "IncorrectValidatorState"

  Register Validator Tests
    ✔ Register validator with 4 operators emits "ValidatorAdded" (78ms)
    ✔ Register validator with 4 operators gas limit (78ms)
    ✔ Register 2 validators into the same cluster gas limit (155ms)
    ✔ Register 2 validators into the same cluster and 1 validator into a new cluster gas limit (228ms)
    ✔ Register 2 validators into the same cluster with one time deposit gas limit (131ms)
    ✔ Register validator with 7 operators gas limit (97ms)
    ✔ Register 2 validators with 7 operators into the same cluster gas limit (187ms)
    ✔ Register 2 validators with 7 operators into the same cluster and 1 validator into a new cluster
with 7 operators gas limit (277ms)
    ✔ Register 2 validators with 7 operators into the same cluster with one time deposit gas limit
(170ms)
    ✔ Register validator with 10 operators gas limit (114ms)
    ✔ Register 2 validators with 10 operators into the same cluster gas limit (216ms)
    ✔ Register 2 validators with 10 operators into the same cluster and 1 validator into a new cluster
with 10 operators gas limit (322ms)
```

```
    ✔ Register 2 validators with 10 operators into the same cluster with one time deposit gas limit
(202ms)
    ✔ Register validator with 13 operators gas limit (135ms)
    ✔ Register 2 validators with 13 operators into the same cluster gas limit (262ms)
    ✔ Register 2 validators with 13 operators into the same cluster and 1 validator into a new cluster
with 13 operators gas limit (382ms)
    ✔ Register 2 validators with 13 operators into the same cluster with one time deposit gas limit
(259ms)
    ✔ Get cluster burn rate (138ms)
    ✔ Get cluster burn rate when one of the operators does not exist
    ✔ Register validator with incorrect input data reverts "IncorrectClusterState" (106ms)
    ✔ Register validator in a new cluster with incorrect input data reverts "IncorrectClusterState"
(44ms)
    ✔ Register validator when an operator does not exist in the cluster reverts "OperatorDoesNotExist"
(49ms)
    ✔ Register validator with a removed operator in the cluster reverts "OperatorDoesNotExist" (50ms)
    ✔ Register cluster with unsorted operators reverts "UnsortedOperatorsList" (43ms)
    ✔ Register cluster with duplicated operators reverts "OperatorsListNotUnique" (61ms)
    ✔ Register validator into a cluster with an invalid amount of operators reverts
"InvalidOperatorIdsLength" (120ms)
    ✔ Register validator with an invalid public key length reverts "InvalidPublicKeyLength"
    ✔ Register validator with not enough balance reverts "InsufficientBalance" (69ms)
    ✔ Register validator in a liquidatable cluster with not enough balance reverts "InsufficientBalance"
(146ms)
    ✔ Register an existing validator with same operators setup reverts "ValidatorAlreadyExists" (43ms)
    ✔ Register an existing validator with different operators setup reverts "ValidatorAlreadyExists"
(44ms)
    ✔ Surpassing max number of validators per operator reverts "ExceedValidatorLimit" (4095ms)
    ✔ Register whitelisted validator in 1 operator with 4 operators emits "ValidatorAdded" (121ms)
    ✔ Register a non whitelisted validator reverts "CallerNotWhitelisted" (102ms)
    ✔ Retrieve an existing validator
    ✔ Retrieve a non-existing validator

  Remove Validator Tests
    ✔ Remove validator emits "ValidatorRemoved" (45ms)
    ✔ Remove validator after cluster liquidation period emits "ValidatorRemoved" (44ms)
    ✔ Remove validator gas limit (4 operators cluster) (44ms)
    ✔ Remove validator gas limit (7 operators cluster) (149ms)
    ✔ Remove validator gas limit (10 operators cluster) (182ms)
    ✔ Remove validator gas limit (13 operators cluster) (223ms)
    ✔ Remove validator with a removed operator in the cluster (71ms)
    ✔ Register a removed validator and remove the same validator again (145ms)
    ✔ Remove validator from a liquidated cluster (84ms)
    ✔ Remove validator with an invalid owner reverts "ValidatorDoesNotExist"
    ✔ Remove validator with an invalid operator setup reverts "IncorrectValidatorState"
    ✔ Remove the same validator twice reverts "ValidatorDoesNotExist" (71ms)


  234 passing (3m)
```

Code Coverage

The code coverage was generated for using `npm run solidity-coverage` and `.solcover.js` provided below.

```
module.exports = {
    skipFiles: ['deprecated', 'test', 'upgrades', 'mocks'],
};
```

We recommend adding more tests to ensure the branch coverage is larger than 90% before deploying the contracts.

Fix Review: During the fix review, the client added 4 more tests for the view function `getNetworkValidatorsCount()` and gas optimization when registering and removing validators for different operator configurations.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	71.88	98.18	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
SSVNetwork.sol	100	80.77	100	100	
SSVNetworkViews.sol	100	33.33	95.45	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/	98.18	85.71	100	94.05	
ClusterLib.sol	100	100	100	100	
CoreLib.sol	88.89	50	100	76.92	15,21,44
OperatorLib.sol	100	90	100	95.65	50
ProtocolLib.sol	100	75	100	91.67	43
SSVStorage.sol	100	100	100	100	
SSVStorageProtocol.sol	100	100	100	100	
Types.sol	100	100	100	100	
ValidatorLib.sol	100	100	100	100	
contracts/modules/	99.55	90.35	97.78	99.68	
SSVClusters.sol	100	85.94	100	100	
SSVDAO.sol	100	100	100	100	
SSVOperators.sol	100	97.22	100	100	
SSVViews.sol	98.18	90	94.74	98.31	203
contracts/token/	0	0	0	0	
SSVToken.sol	0	0	0	0	21
All files	99.1	85.26	96.88	98.5	

Changelog

- 2023-10-30 - Initial report
- 2023-11-08 - Final report

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