Smart Contract Security Audit Report

Symbiosis Staking Update

1. Contents

2. General Information

This report contains information about the results of the security audit of the Symbiosis (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 14/07/2025 to 18/07/2025.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/symbiosis-finance/pos-contracts. Initial review was done for the commit f2d6fe824aea03d1db32ed832eb965f807851c4d, and the re-testing was done for the commit 0ee52fa85fcbed855f09664c32a5766a4d7a2403.

The following contracts have been tested:

- contracts/Validator.sol
- contracts/utils/StakingView.sol
- contracts/utils/EventLogger.sol
- contracts/utils/RewardCounter.sol
- contracts/SymbiToken.sol
- contracts/Staking.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a single high exploitable security issue which has been fixed and re-tested in the course of the work.

The other suggestions included fixing the medium and low risk issues and some best practices (see Security Process Improvement).

The Symbiosis team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 12, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Exit function in Validator contract does not account for stakerToExitAmount	contracts/Validator.sol	**High**	Fixed

Issue	Contract	Risk Level	Status
Delegated stake withdraw may work as not expected	contracts/Validator.sol	**Medium**	Fixed
DoS in Staking contract	contracts/Staking.sol	**Medium**	Fixed
Jailed validators still have right to vote	contracts/Staking.sol	**Medium**	Fixed
Validators may receive fewer rewards than they should	contracts/Staking.sol	**Medium**	Fixed
Epoch change may be blocked by exiting validators	contracts/Staking.sol	**Medium**	Fixed
changeEpoch manipulation	contracts/Staking.sol	**Medium**	Fixed
Potential validator slot griefing	contracts/Staking.sol	**Medium**	Fixed
Ineffective check in prepareWithdrawDelegatedStake()	contracts/Validator.sol	**Low**	Fixed
Allowed fees/rates should be capped by smart contracts	contracts/utils/RewardCounter.sol	**Low**	Fixed
Use `abi.encodeCall()` instead of `abi.encodeSelector`	contracts/Staking.sol	**Low**	Fixed
Use Ownable2Step rather than Ownable for transfers of ownership	contracts/Staking.sol contracts/SymbiToken.sol contracts/utils/RewardCounter.sol contracts/utils/StakingView.sol	**Low**	Fixed
Lack of delegateStakeThreshold check	contracts/Staking.sol	**Low**	Fixed

Issue	Contract	Risk Level	Status
Validators may lose their reward	contracts/Staking.sol	Low	Acknowledged
Check at least 1 veto user exists	Staking.sol	Low	Fixed
Missing Index Validation Allows totalStaked Inflation	contracts/Staking.sol	Low	Fixed
Potential wrong element removal in `_removeRelayer()`	contracts/Staking.sol	Info	Acknowledged
Private and internal variables should start with an underscore	contracts/utils/StakingView.sol	Info	Fixed
Contract does not follow the Solidity style guide's suggested layout ordering	contracts/utils/EventLogger.sol	Info	Fixed
Function ordering does not follow the Solidity style guide	contracts/Staking.sol contracts/SymbiToken.sol contracts/Validator.sol contracts/interfaces/IValidator.sol contracts/utils/RewardCounter.sol contracts/utils/StakingView.sol	Info	Fixed
Typos in the code	contracts/Validator.sol	Info	Fixed
Use time units for readability	contracts/utils/RewardCounter.sol	Info	Fixed
Missing license	contracts/utils/EventLogger.sol	Info	Fixed
Events that mark critical parameter changes should	contracts/Staking.sol	Info	Acknowledged

Issue	Contract	Risk Level	Status
contain both the old and the new value			
Setters should have initial value check	contracts/Staking.sol contracts/utils/RewardCounter.sol	Info	Acknowledged
Use != 0 instead of > 0 for unsigned integer comparison	contracts/Staking.sol	Info	Fixed
Avoid mutating function parameters	contracts/Staking.sol contracts/Validator.sol	Info	Acknowledged
Consider using `delete` rather than assigning zero to clear values	contracts/Staking.sol	Info	Fixed
Change `public` to `external` for functions that are not called internally	contracts/Staking.sol	Info	Fixed
Unused event definition	contracts/Staking.sol	Info	Fixed
Consider disabling `renounceOwnership()`	contracts/Staking.sol contracts/utils/RewardCounter.sol	Info	Acknowledged
NatSpec `@param` is missing	contracts/Staking.sol contracts/Validator.sol	Info	Fixed
NatSpec `@return` is missing	contracts/Staking.sol contracts/Validator.sol	Info	Fixed
Using named parameters in mapping is best practice	contracts/Staking.sol contracts/Validator.sol contracts/utils/EventLogger.sol	Info	Acknowledged
Redundant Hardhat Console Import	pos-contracts/contracts/Staking.sol	Info	Fixed

Issue	Contract	Risk Level	Status
Duplicate functions in Validator contract	contracts/Validator.sol	**Info**	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Exit function in Validator contract does not account for stakerToExitAmount

Risk Level: High

Status: Fixed

Contracts:

- contracts/Validator.sol

Location: Function: `exit()`.

Description:

The exit function in the Validator contract does not properly account for the `stakerToExitAmount` variable. As a result, if a validator has a nonzero `stakerToExitAmount`, the exit function may not work as intended.

Specifically, if `stakerToExitAmount > 0`, calling exit with `amount == 0` will not delete the validator. This is because the logic checks if (`amount == stakerAmount`), but `stakerAmount` is only updated in `updateAmounts()`. Thus, when `stakerToExitAmount == stakerAmount`, a call with `amount == 0` does not trigger validator elimination, and the validator remains active.

This can lead to a situation where a validator withdraw part of their stake (increasing `stakerToExitAmount`), then withdraws the remainder (making `stakerToExitAmount == stakerAmount`), and finally calls `exit(0)`. Since the elimination condition is not met, the validator is not removed, and delegators' funds may be locked indefinitely due to the `isActive` check in `prepareWithdrawDelegatedStake`. Because validator was removed from Staking contract `prepareWithdrawDelegatedStake` call will fail not allowing delegators to receive their funds.

```
    function exit(uint256 amount) external onlyValidator onlyActive {
        require(stakerAmount >= amount, "exit: Not enough tokens");
        require(stakerToEnterAmount == 0, "exit: There are toEnter tokens");
        if (amount == 0){
            amount = stakerAmount - stakerToExitAmount;
        }
        require(stakerAmount - amount - stakerToExitAmount >=
IStaking(staking).stakeThreshold() || stakerAmount - stakerToExitAmount ==
amount , "exit: you must leave at least threshold");
        stakerExitEpoch = IStaking(staking).currentEpoch();
        if (amount == stakerAmount) {
            IStaking(staking).exit(validatorId, 0);
            stakerToExitAmount += stakerAmount;
            _eliminateValidator();
        } else {
            IStaking(staking).exit(validatorId, amount);
            stakerToExitAmount += amount;
        }
        eventLogger.logStakerExited(address(this), staker, amount,
IStaking(staking).currentEpoch());
    }
```

Remediation:

Consider counting the `stakerToExitAmount` during the comparison of `stakerAmount`:

```
-- if (amount == stakerAmount) {
++ if (amount == stakerAmount - stakerToExitAmount) {
            IStaking(staking).exit(validatorId, 0);
            stakerToExitAmount += stakerAmount;
            _eliminateValidator();
        } else {
```

5.2. Delegated stake withdraw may work as not expected

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Validator.sol

Location: Function: `prepareWithdrawDelegatedStake`.

Description:

Users can call `prepareWithdrawDelegatedStake` function with a desired amount. The function checks if the delegator has sufficient funds, then deducts the amount from their amount balance and records it in their `toExit` value:

```
delegators[msg.sender].amount -= amount;
delegators[msg.sender].toExit += amount;
```

However, if a user first requests to withdraw a certain amount, and then later decides to fully withdraw by passing `amount == 0`, the following if statement is triggered:

```
if (amount == 0) {
    amount =
        delegators[msg.sender].amount -
        delegators[msg.sender].toExit;
}
```

In this case, instead of withdrawing the full originally intended amount, the user will end up withdrawing only `amount - toExit`.

This may lead to confusion and reduced usability.

Remediation:

Consider modifying the if statement to:

```
if (amount == 0) {
    amount =
        delegators[msg.sender].amount
-       - delegators[msg.sender].toExit;
}
```

5.3. DoS in Staking contract

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `initialize() enter()`.

Description:

In the `Staking` contract the `eventLogger` state variable is never initialized in the `initialize()` function, however this variable is required for the `enter()` function to work. This causes all validator entry attempts to revert, completely disabling the staking functionality.

Missing initialization in `initialize()` function:

```
function initialize(
    address _token,
    IRewardCounter _rewardCounter,
    address _validatorBeacon
) public virtual initializer {
    __Ownable_init();
    token = _token;
    rewardCounter = _rewardCounter;
    indexCounter = 1;
    previousChangeEpochTimestamp = 0;
    validatorBeacon = _validatorBeacon;
    // @audit eventLogger is never initialized (remains address(0))
}
```

Usage in `enter()` function that will revert:

```
function enter(address _user, uint256 _amount, address _relayer) external {
    // ... validation logic ...
    address validator = address(new BeaconProxy(validatorBeacon, ...));

    IEventLogger(eventLogger).addValidator(validator);           // ←
eventLogger = address(0), will revert
    IEventLogger(eventLogger).logStakerEntered(validator, _user, _amount,
currentEpoch); // ← Will revert

    // ... rest of function never executes
}
```

The contract can be set via `setEventLogger()` by the owner, but the production deployment should not contain this function, as it stated in the @dev comment:

```
/**
* @notice setter for the new event logger contract
* @dev Must be deleted in prod
*/
function setEventLogger(address _eventLogger) external onlyOwner {
    eventLogger = _eventLogger;
    emit EventLoggerSet(_eventLogger);
}
```

Remediation:

Consider adding `EventLogger` parameter to the `initialize` function and properly initialize the state variable:

```
function initialize(
    address _token,
    IRewardCounter _rewardCounter,
    address _validatorBeacon,
    address _eventLogger   // ← Add missing parameter
) public virtual initializer {
    __Ownable_init();
    token = _token;
    rewardCounter = _rewardCounter;
    indexCounter = 1;
    previousChangeEpochTimestamp = 0;
    validatorBeacon = _validatorBeacon;
    eventLogger = _eventLogger;  // ← Initialize the state variable
}
```

5.4. Jailed validators still have right to vote

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: changeEpoch.

Description:

Jailed validators can still participate in epoch consensus by signing epoch changes. The `isJailed[stake[index].validator]` check is only present during reward calculations (line 494) but missing during the signature verification loop (line 421), allowing jailed validators to contribute to veto thresholds, MPC group verification, and stake majority requirements. This creates an inconsistent security mechanism where jailing doesn't fully prevent malicious validator participation in governance.

```
for (uint256 i = 0; i < sigs.length; ++i) {
    // ... signature recovery ...
    uint256 index = relayerToIndex[relayer]; // Line 421: No jail check
    require(index != 0, "Signer cannot sign");

    if (isVeto[stake[index].validator]) {
        signedVetoAndMpc[0]++; // Jailed veto validators still count
    }
    totalStakedPower += validatorAmount; // Jailed stake counts toward
majority
}
```

Jail check only in reward distribution:

```
if (!isJailed[stake[index].validator] && stake[index].toExit <
stake[index].amount) {
    // Reward distribution - jailed validators properly excluded
}
```

Remediation:

Consider adding jail status verification in the signature verification loop to prevent jailed validators from participating in consensus.

5.5. Validators may receive fewer rewards than they should

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `changeEpoch()`.

Description:

If the relayer of a validator who is fully exiting signs the epoch change, the validator's funds are still included in `totalStakedPower`. Since validators who are exiting with their entire stake do not receive rewards, and rewards are calculated based on `totalStakedPower`, a portion of the rewards for the epoch change will remain undistributed. This means that active validators will receive less rewards than they should.

```
...
uint256 validatorAmount = stake[index].amount;
totalStakedPower += validatorAmount;
...

...
if (totalStaked > 0) {
    for (uint256 i = 0; i < sigs.length; ++i) {
        address relayer = ECDSA.recover(
            getSignatureBaseForChangeEpoch(mpcAddress, mpcGroup),
            sigs[i]
        );
        uint256 index = relayerToIndex[relayer];
        if (
            !isJailed[stake[index].validator] &&
        stake[index].toExit < stake[index].amount // @audit you do not receive
a reward if you exit with your full stake
        ) {
            uint256 stakerAmount = IValidator(stake[index].validator)
            .stakerAmount();
            (
            uint256 validatorReward,
            uint256 totalStakerReward
            ) = rewardCounter.calculateReward(
                totalStakedPower, // @audit validators receive less rewards
than they should
                stake[index].amount,
                stakerAmount,
                previousChangeEpochTimestamp
            );
            totalStaked += validatorReward;
            stake[index].amount += validatorReward;
            IValidator(stake[index].validator).distributeReward(
                validatorReward,
                totalStakerReward,
                currentEpoch
            );
        }
    }
...
```

Remediation:

Consider ensuring that validators who are fully exiting are not included in the calculation of `totalStakedPower`.

```
+                           if (stake[index].toExit < stake[index].amount) {
+                   uint256 validatorAmount = stake[index].amount;
+                 } else {
+                     uint256 validatorAmount = 0;
+                 }

             totalStakedPower += validatorAmount;
```

5.6. Epoch change may be blocked by exiting validators

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `getActiveSignersTotalStake()`.

Description:

If a validator fully exits, their funds are still included in the getActiveSignersTotalStake calculation until the epoch is changed. If this exiting validator holds a large amount of stake (for example, 1/3 of the total), and does not sign the epoch change (because their relayer is no longer active), the epoch change can be blocked. This is because the required threshold for changing the epoch is calculated based on the total active stake, which still includes the exiting validator's funds. As a result, the remaining active validators may not be able to reach the required 2/3 majority, preventing the epoch from being updated.

```
function getActiveSignersTotalStake() public view returns (uint256) {
    uint256 activeTotalStake = 0;
    for (uint256 i = 0; i < relayers.length; i++) {
        uint256 validatorId = relayerToIndex[relayers[i]];
        if (!isJailed[stake[validatorId].validator]) {
            activeTotalStake += stake[validatorId].amount; // @audit includes 
fully exiting validators
        }
    }
    return activeTotalStake;
}
function changeEpoch(
        address mpcAddress,
        address[] memory mpcGroup,
        bytes[] calldata sigs
    ) external {
    ...

    require(
        totalStakedPower >= (getActiveSignersTotalStake() * 2) / 3 + 1 ||
        msg.sender == currentForceEpochChanger,
        "non-majority"
    ); // @audit  this may prevent the epoch from being changed
    ...
```

Remediation:

Consider excluding fully exiting validators from the `getActiveSignersTotalStake` calculation, so that only truly active validators are counted towards the threshold for epoch change.

5.7. changeEpoch manipulation

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `exit, changeEpoch`.

Description:

When users calls `exit` theirs tokens from `stake[index].toEnter` are assigned to `stake[index].amount`. In case `withdrawDelay` is set to 0 and is part of an mpc group, they can do the following attack:

- take flash loan
- enter
- exit
- call change epoch
- withdraw

As a result, they will become an mpc leader and some of the rewards will be gone, because of incorrect totalStakedPower.

Remediation:

Consider not counting funds that are added during the `exit(0)` call like it's done in distribute rewards part of `changeEpoch()`

```
+                        if (stake[index].toExit < stake[index].amount) {
+                uint256 validatorAmount = stake[index].amount;
+              } else {
+                  uint256 validatorAmount = 0;
+              }

           totalStakedPower += validatorAmount;
```

5.8. Potential validator slot griefing

Risk Level: Medium

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `setStakeThreshold`.

Description:

An attacker may front-run the `setStakeThreshold` function and fill all possible slots for validators with zero stake amounts. This action can prevent legitimate users from staking their funds in the protocol and participating as validators.

Remediation:

Consider setting `stakeThreshold` in initialise function.

5.9. Ineffective check in prepareWithdrawDelegatedStake()

Risk Level: Low

Status: Fixed

Contracts:

- contracts/Validator.sol

Location: Lines: 285. Function: `prepareWithdrawDelegatedStake()`.

Description:

The check `delegators[msg.sender].lastRestakeEpoch <= IStaking(staking).currentEpoch()` is always true and provides no protection. Since `lastRestakeEpoch` is set to `currentEpoch()` during delegation/restaking, and epochs only increase, this condition will always pass.

Remediation:

Remove the ineffective check or implement proper logic with a strict check if a delay mechanism is intended:

```
// Fixed version:
require(delegators[msg.sender].lastRestakeEpoch <
IStaking(staking).currentEpoch(), "prepareWithdrawDelegatedStake: ToEnter
delay");
```

5.10. Allowed fees/rates should be capped by smart contracts

Risk Level: Low

Status: Fixed

Contracts:

- contracts/utils/RewardCounter.sol

Description:

Fees/rates should be required to be below 100%, preferably at a much lower limit, to ensure users don’t have to monitor the blockchain for changes prior to using the protocol.

```
File: contracts/utils/RewardCounter.sol

    function setFixedRewardRate(uint256 _newValue) external onlyOwner {
        fixedRewardRate = _newValue;
    }
```

Remediation:

Implement a check in `RewardCounter.sol` to ensure that the `fixedRewardRate` is below a specified maximum value before setting it.

5.11. Use `abi.encodeCall()` instead of `abi.encodeSelector`

Risk Level: Low

Status: Fixed

Contracts:

- contracts/Staking.sol

Description:

`abi.encodeCall()` has compiler type safety.

```
File: contracts/Staking.sol

        address validator = address(new BeaconProxy(
                validatorBeacon,
                abi.encodeWithSelector(bytes4(0x001a23d5), _user, index,
address(this), token, _amount, eventLogger)
            ));
```

Remediation:

Replace `abi.encodeWithSelector` with `abi.encodeCall` in `Staking.sol` to ensure type safety during encoding.

5.12. Use Ownable2Step rather than Ownable for transfers of ownership

Risk Level: Low

Status: Fixed

Contracts:

- contracts/Staking.sol
- contracts/SymbiToken.sol
- contracts/utils/RewardCounter.sol
- contracts/utils/StakingView.sol

Description:

Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
File: contracts/Staking.sol

19: contract Staking is OwnableUpgradeable {
File: contracts/SymbiToken.sol

9: contract SymbiToken is ERC20Burnable, ERC20VotesComp, Ownable {
File: contracts/utils/RewardCounter.sol

10: contract RewardCounter is OwnableUpgradeable, IRewardCounter {
File: contracts/utils/StakingView.sol

10: contract StakingView is OwnableUpgradeable, IStakingView {
```

Remediation:

Replace instances of `Ownable` with `Ownable2Step` or `Ownable2StepUpgradeable` in the contracts `Staking.sol`, `SymbiToken.sol`, `RewardCounter.sol`, and `StakingView.sol`.

5.13. Lack of delegateStakeThreshold check

Risk Level: Low

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `prepareWithdrawDelegatedStake()`.

Description:

When staking, users are required to deposit an amount greater than delegateStakeThreshold. However, when preparing to withdraw, there is no check to ensure that the remaining amount after withdrawal is still above the delegateStakeThreshold. As a result, a delegator can leave any amount in their account and still remain an active delegator, even if their stake falls below the required threshold.

```
function prepareWithdrawDelegatedStake(uint256 amount) external {
        ...
        // @audit missing check for delegateStakeThreshold
        require(
            delegators[msg.sender].amount >= amount,
            "prepareWithdrawDelegatedStake: You dont have enough stake"
        );
        require(amount != 0, 'Nothing to withdraw');
        ...
```

Remediation:

Consider ensuring that after preparing a withdrawal, the remaining stake of the delegator is not less than delegateStakeThreshold.

5.14. Validators may lose their reward

Risk Level: Low

Status: This kind of behavior is supposed to be punished through jailing at the moment.

Contracts:

- contracts/Staking.sol

Location: Function: `changeEpoch()`.

Description:

The absence of access control in the changeEpoch() function allows any user to arbitrarily modify the signatures array, removing relayers until the condition totalStakedPower >= (getActiveSignersTotalStake() * 2) / 3 + 1 is met. This enables a malicious actor to exclude certain validators from the epoch change, causing them to miss out on their rewards.

```
if (totalStaked > 0) {
    // @audit-issue Only relayers present in sigs receive rewards. If all 
relayers sign the epoch change,
    // someone can remove 1/3 of the relayers so that the condition passes,
    // but those removed do not receive rewards.
    for (uint256 i = 0; i < sigs.length; ++i) {
        address relayer = ECDSA.recover(
            getSignatureBaseForChangeEpoch(mpcAddress, mpcGroup),
            sigs[i]
        );
        uint256 index = relayerToIndex[relayer];
        if (
            !isJailed[stake[index].validator] &&
        stake[index].toExit < stake[index].amount
        ) {
            uint256 stakerAmount = IValidator(stake[index].validator)
            .stakerAmount();
            (
            uint256 validatorReward,
            uint256 totalStakerReward
            ) = rewardCounter.calculateReward(
                totalStakedPower,
                stake[index].amount,
                stakerAmount,
                previousChangeEpochTimestamp
            );
            totalStaked += validatorReward;
            stake[index].amount += validatorReward;
            IValidator(stake[index].validator).distributeReward(
                validatorReward,
                totalStakerReward,
                currentEpoch
            );
        }
    }
}
```

Remediation:

Consider adding access control to the changeEpoch() function.

5.15. Check at least 1 veto user exists

Risk Level: Low

Status: Fixed

Contracts:

- Staking.sol

Location: Function: `setVeto()`.

Description:

The `setVeto()` function allows the contract owner to change the veto status of validators. However, there is no check to ensure that at least one veto user always exists. If all users are removed from the veto list (i.e., vetoCount becomes zero), this could potentially break functionality of changing the epoch that relies on the existence of at least one veto user.

```
function setVeto(address validator, bool status) external onlyOwner {
        if (status == true) {
            if (isVeto[validator] != true) {
                vetoCount++;
            }
        } else {
            if (isVeto[validator] != false) {
                vetoCount--;
            }
        }
        isVeto[validator] = status;
        emit VetoSet(validator, status);
    }
```

Remediation:

Consider checking that `vetoCount` is at least 1.

5.16. Missing Index Validation Allows totalStaked Inflation

Risk Level: Low

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: `restake() exit()`.

Description:

The `restake` and `exit` functions in the Staking contract lack proper validation for the `index` parameter, specifically missing a check for `index != 0`. This vulnerability allows malicious actors to inflate the `totalStaked` variable through a donation.

```
require(validatorToIndex[msg.sender] == index, "it is not your index");
```

When `index = 0` is passed, the mapping `validatorToIndex[msg.sender]` returns 0 (the default value for uninitialized mappings), making the condition `validatorToIndex[msg.sender] == 0` evaluate to true for any address that hasn't been registered as a validator. This bypasses the access control check and allows unauthorized addresses to interact with the functions.

In the `restake` function, when `index = 0` is passed: 1. The access control check passes for any unregistered address 2. The function transfers tokens from `msg.sender` to the contract 3. The tokens are added to `stake[0].toEnter`

In the `exit` function with `amount = 0`: 1. The access control check passes for any unregistered address 2. The function calculates the total amount from `stake[0]` (where `toEnter` was previously increased via `restake`) 3. It adds `stake[0].toEnter` to `totalStaked` directly 4. This inflates `totalStaked` by the amount that was donated in the previous `restake` call

Remediation:

Add explicit validation to ensure `index != 0` in both functions:

```
require(index != 0, "Invalid index");
```

5.17. Potential wrong element removal in `_removeRelayer()`

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Staking.sol

Location: Function: `_removeRelayer()`.

Description:

The function `_removeRelayer()` could remove the first relayer instead of the target if `signerToDelete` is not found in the array. However, this scenario cannot occur now since this function is only called after successful validation.

Remediation:

Consider adding safety check to `_removeRelayer()`:

```
require(found, "Relayer not found");
```

5.18. Private and internal variables should start with an underscore

Risk Level: Info

Status: Fixed

Contracts:

- contracts/utils/StakingView.sol

Description:

Variables are not following solidity style guide:

```
File: contracts/utils/StakingView.sol
12:     IStaking staking;
13:     IRewardCounter rewardCounter;
```

Remediation:

Rename the `staking` and `rewardCounter` variables in `StakingView.sol` to `_staking` and `_rewardCounter` to follow the style guide.

5.19. Contract does not follow the Solidity style guide's suggested layout ordering

Risk Level: Info

Status: Fixed

Contracts:

- contracts/utils/EventLogger.sol

Description:

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Errors
5. Modifiers
6. Functions

Source: https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-layout

```
File: contracts/utils/EventLogger.sol

19:     /// @audit Event definition can not go after modifier definition
20:     event ValidatorAdded(address indexed validator);
```

Remediation:

Reorder the elements in `EventLogger.sol` to follow the recommended order: type declarations, state variables, events, errors, modifiers, and functions.

5.20. Function ordering does not follow the Solidity style guide

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol
- contracts/SymbiToken.sol
- contracts/Validator.sol
- contracts/interfaces/IValidator.sol
- contracts/utils/RewardCounter.sol

- contracts/utils/StakingView.sol

Description:

Current code doesn't follow the recommended order: constructor, receive function (if exists), fallback function (if exists), external, public, internal, and private functions.

```
File: contracts/Staking.sol

168:     /// @audit External view function can not go after public view
function
169:     function getMpcGroupByEpoch(

File: contracts/SymbiToken.sol

34:     /// @audit External function can not go after internal function
35:     function transferBatch(address[] memory to, uint256[] memory amounts)
external {

File: contracts/Validator.sol

117:     /// @audit External view function can not go after public function
118:     function getDelegatorRewards(address _delegator) external view
returns (uint256) {

File: contracts/interfaces/IValidator.sol

6:     /// @audit External function can not go after external view function
7:     function distributeReward(uint256 validatorReward, uint256
stakerReward, uint256 epoch) external;

File: contracts/utils/RewardCounter.sol

23:     /// @audit External function can not go after public function
24:     function setFixedRewardRate(uint256 _newValue) external onlyOwner {

File: contracts/utils/StakingView.sol

26:     /// @audit External view function can not go after public function
27:     function getLeaderByEpoch(uint256 epoch) external view returns
(address) {
```

Remediation:

Reorder the functions in `Staking.sol`, `SymbiToken.sol`, `Validator.sol`, `IValidator.sol`, `RewardCounter.sol`, and `StakingView.sol` to follow the recommended order: constructor, receive function (if exists), fallback function (if exists), external, public, internal, and private functions.

References:

- https://docs.soliditylang.org/en/latest/style-guide.html#order-of-layout

5.21. Typos in the code

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Validator.sol

Description:

There is a typo in the code:

```
File: contracts/Validator.sol

312:      /// @audit delegator
313:      * @notice Withdraw tokens that are allocated for withdraw by
delgator
```

Remediation:

Consider correcting the typo.

5.22. Use time units for readability

Risk Level: Info

Status: Fixed

Contracts:

- contracts/utils/RewardCounter.sol

Description:

The Solidity documentation provides units for seconds, minutes, hours, days, and weeks, which are already defined and available for use. It is recommended to utilize these predefined units when working with time-related calculations in Solidity.

```
File: contracts/utils/RewardCounter.sol

20:         epochTime = 60 * 60 * 24 * 7;
```

Remediation:

Replace the hardcoded time calculation in `RewardCounter.sol` with the appropriate time unit, such as `7 days`.

5.23. Missing license

Risk Level: Info

Status: Fixed

Contracts:

- contracts/utils/EventLogger.sol

Description:

Consider adding a license to the file.

```
File: contracts/utils/EventLogger.sol

1: pragma solidity 0.8.19;
```

Remediation:

Add an SPDX license identifier at the top of `EventLogger.sol` to specify the license under which the code is distributed.

5.24. Events that mark critical parameter changes should contain both the old and the new value

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Staking.sol

Description:

It is important to have both the old and the new value in the event. This way, the user can easily track the changes.

```
File: contracts/Staking.sol
```

```
569:          emit EpochSet(epoch);

578:          emit EventLoggerSet(_eventLogger);

602:          emit WithdrawDelaySet(_withdrawDelay);

619:          emit VetoSet(validator, status);

633:          emit JailerSet(_jailer);

650:          emit EpochChangerSet(_epochChanger);

675:          emit ValidatorBeaconSet(newValidatorBeacon);
```

Remediation:

Update the events in `Staking.sol` to include both the old and new values for the specified instances.

5.25. Setters should have initial value check

Risk Level: Info

Status: partially fixed

Contracts:

- contracts/Staking.sol
- contracts/utils/RewardCounter.sol

Description:

Setters should have initial value check to prevent assigning wrong value to the variable. Assignment of wrong value can lead to unexpected behavior of the contract.

```
File: contracts/Staking.sol

566:      /// @audit Not validated: epoch

567:      function setEpoch(uint256 epoch) external onlyOwner {
568:          currentEpoch = epoch;
569:          emit EpochSet(epoch);
570:      }

575:      /// @audit Not validated: _eventLogger
```

```
576:     function setEventLogger(address _eventLogger) external onlyOwner {
577:         eventLogger = _eventLogger;
578:         emit EventLoggerSet(_eventLogger);
579:     }

583:     /// @audit Not validated: _newThreshold

584:     function setStakeThreshold(uint256 _newThreshold) external onlyOwner
{
585:         stakeThreshold = _newThreshold;
586:         emit StakeThresholdSet(_newThreshold);
587:     }

591:     /// @audit Not validated: _newThreshold

592:     function setStakeNumberThreshold(uint256 _newThreshold) external
onlyOwner {
593:         stakeNumberThreshold = _newThreshold;
594:         emit StakeNumberThresholdSet(_newThreshold);
595:     }

599:     /// @audit Not validated: _withdrawDelay

600:     function setWithdrawDelay(uint256 _withdrawDelay) external onlyOwner
{
601:         withdrawDelay = _withdrawDelay;
602:         emit WithdrawDelaySet(_withdrawDelay);
603:     }

672:     /// @audit Not validated: newValidatorBeacon

673:     function setValidatorBeacon(address newValidatorBeacon) external
onlyOwner {
674:         validatorBeacon = newValidatorBeacon;
675:         emit ValidatorBeaconSet(newValidatorBeacon);
676:     }
File: contracts/utils/RewardCounter.sol

23:     /// @audit Not validated: _newValue

24:     function setFixedRewardRate(uint256 _newValue) external onlyOwner {
25:         fixedRewardRate = _newValue;
26:     }

27:     /// @audit Not validated: _newValue

28:     function setRewardPerEpoch(uint256 _newValue) external onlyOwner {
29:         rewardPerEpoch = _newValue;
30:     }
```

```
31:     /// @audit Not validated: _newValue

32:     function setEpochTime(uint256 _newValue) external onlyOwner {
33:         epochTime = _newValue;
34:     }
```

Remediation:

Add checks in the setter functions in `Staking.sol` and `RewardCounter.sol` to ensure that the new value is different from the current value before assignment.

5.26. Use != 0 instead of > 0 for unsigned integer comparison

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol

Location: Function: <unknown>.

Description:

The code is using the != 0 instead of > 0 for unsigned integer comparison:

```
File: contracts/Staking.sol

483:         if (totalStaked > 0) {

720:         for (; i > 0; --i) {
```

Remediation:

Consider replacing instances of `> 0` with `!= 0` in `Staking.sol` for the specified lines to improve code clarity and correctness.

5.27. Avoid mutating function parameters

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Staking.sol
- contracts/Validator.sol

Location: Function: <unknown>.

Description:

Function parameters in Solidity are passed by value, meaning they are essentially local copies. Mutating them can lead to confusion and errors because the changes don’t persist outside the function. By keeping function parameters immutable, you ensure clarity in code behavior, preventing unintended side-effects. If you need to modify a value based on a parameter, use a local variable inside the function, leaving the original parameter unaltered. By adhering to this practice, you maintain a clear distinction between input data and the internal processing logic, improving code readability and reducing the potential for bugs.

```
File: contracts/Staking.sol

327:     function exit(uint256 index, uint256 amount) external {
File: contracts/Validator.sol

161:     function exit(uint256 amount) external onlyValidator onlyActive {

279:     function prepareWithdrawDelegatedStake(uint256 amount) external {
```

Remediation:

Use local variables instead of mutating function parameters in `Staking.sol` and `Validator.sol` to maintain clarity and prevent unintended side-effects.

5.28. Consider using `delete` rather than assigning zero to clear values

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol

Description:

The `delete` keyword more closely matches the semantics of what is being done, and draws more attention to the changing of state, which may lead to a more thorough audit of its associated logic.

```
File: contracts/Staking.sol

309:          relayerToIndex[oldRelayer] = 0;

366:          allocateToWithdraw[sender] = 0;

523:                  stake[index].toEnter = 0;

530:                  stake[index].toExit = 0;

533:                      relayerToIndex[stake[index].relayer] = 0;

534:                      validatorToIndex[stake[index].validator] = 0;

535:                      stakerToValidator[staker] = address(0);

537:                      stake[index].validator = address(0);

538:                      stake[index].relayer = address(0);
```

Remediation:

Replace assignments of zero with the `delete` keyword in `Staking.sol` for the specified instances to improve code clarity.

5.29. Change `public` to `external` for functions that are not called internally

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol

Description:

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

```
File: contracts/Staking.sol

162:     function getRelayersLength() public view returns (uint256) {
```

Remediation:

Change the visibility of the `getRelayersLength` function in `Staking.sol` from `public` to `external`.

5.30. Unused event definition

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol

Description:

The following events are never emitted, consider to remove them.

```
File: contracts/Staking.sol

93:     event RewardsAdded(
```

Remediation:

Remove the `RewardsAdded` event definition from `Staking.sol` if it is not used anywhere in the contract.

5.31. Consider disabling `renounceOwnership()`

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Staking.sol
- contracts/utils/RewardCounter.sol

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities. The OpenZeppelin's Ownable used in this project contract implements renounceOwnership. This can represent a certain risk if the ownership is renounced for any other reason than by design. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.

```
File: contracts/Staking.sol

19: contract Staking is OwnableUpgradeable {
File: contracts/utils/RewardCounter.sol

10: contract RewardCounter is OwnableUpgradeable, IRewardCounter {
```

Remediation:

Override the `renounceOwnership` function in `Staking.sol` and `RewardCounter.sol` to disable it, ensuring that ownership cannot be renounced unintentionally.

5.32. NatSpec `@param` is missing

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol
- contracts/Validator.sol

Location: Function: <unknown>.

Description:

The NatSpec `@param` is missing:

```
File: contracts/Staking.sol

169: function getMpcGroupByEpoch(
        uint256 epoch
    ) external view returns (address[] memory)

178: function getSignatureBaseForChangeEpoch(
        address mpcAddress,
        address[] memory mpcGroup
    ) public view returns (bytes32)

567: function setEpoch(uint256 epoch) external onlyOwner

576: function setEventLogger(address _eventLogger) external onlyOwner

584: function setStakeThreshold(uint256 _newThreshold) external onlyOwner

592: function setStakeNumberThreshold(uint256 _newThreshold) external
onlyOwner

600: function setWithdrawDelay(uint256 _withdrawDelay) external onlyOwner

608: function setVeto(address validator, bool status) external onlyOwner

625: function setJailer(address _jailer) external

639: function setEpochChanger(address _epochChanger) external

656: function setForceEpochChanger(address _forceEpochChanger) external

673: function setValidatorBeacon(address newValidatorBeacon) external
onlyOwner

683: function setJailedStatus(address _validator, bool _status) external
```

Another example:

```
File: contracts/Validator.sol

118: function getDelegatorRewards(address _delegator) external view returns
(uint256)

376: function distributeReward(
        uint256 validatorReward,
        uint256 stakerReward,
        uint256 epoch
    ) external onlyStaking

429: function _claimReward(address _delegator) internal
```

Remediation:

Add the `@param` tag to the NatSpec comments for the functions listed in `Staking.sol` and `Validator.sol` to provide clear documentation of the parameters.

5.33. NatSpec `@return` is missing

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Staking.sol
- contracts/Validator.sol

Description:

The NatSpec tag `@return` is missing:

```
File: contracts/Staking.sol

162: function getRelayersLength() public view returns (uint256)

169: function getMpcGroupByEpoch(
        uint256 epoch
    ) external view returns (address[] memory)

178: function getSignatureBaseForChangeEpoch(
        address mpcAddress,
        address[] memory mpcGroup
    ) public view returns (bytes32)

197: function getVetoGroupThreshold() public view returns (uint256)

207: function getActiveSignersTotalStake() public view returns (uint256)

363: function withdraw() external returns (uint256)
File: contracts/Validator.sol

118: function getDelegatorRewards(address _delegator) external view returns
(uint256)
```

Remediation:

Add the `@return` tag to the NatSpec comments for the functions listed in `Staking.sol` and `Validator.sol` to provide clear documentation of the return values.

5.34. Using named parameters in mapping is best practice

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/Staking.sol
- contracts/Validator.sol
- contracts/utils/EventLogger.sol

Description:

The mappings in `Staking.sol`, `Validator.sol`, and `EventLogger.sol` are not named:

```
File: contracts/Staking.sol

30:     mapping(uint256 => Node) public stake;

37:     mapping(address => bool) public wasJailer;

40:     mapping(address => bool) public wasEpochChanger;

43:     mapping(address => bool) public wasForceEpochChanger;

46:     mapping(address => uint256) public allocateToWithdraw;

47:     mapping(address => bool) public isJailed;

56:     mapping(address => uint256) public validatorToIndex;

57:     mapping(address => uint256) public relayerToIndex;

58:     mapping(address => bool) public isVeto;

60:     mapping(address => address) public stakerToValidator;

61:     mapping(address => address[]) public mpcToMpcGroup;

62:     mapping(address => address) public mpcToLeader;

63:     mapping(uint256 => address) public epochToMpc;
File: contracts/Validator.sol

48:     mapping(address => Delegator) public delegators;

49:     mapping(address => uint256) public delegatorToIndex;

50:     mapping(uint256 => address) public indexToDelegator;

51:     mapping(uint256 => uint256) public epochToRewardPerSIS;
```

```
File: contracts/utils/EventLogger.sol

7:     mapping(address => bool) public isValidator;
```

Remediation:

Ensure that all mappings in `Staking.sol`, `Validator.sol`, and `EventLogger.sol` use named parameters for clarity.

5.35. Redundant Hardhat Console Import

Risk Level: Info

Status: Fixed

Contracts:

- pos-contracts/contracts/Staking.sol

Description:

The contract imports `hardhat/console.sol` which is only used for debugging purposes during development.

Remediation:

Remove the redundant import.

5.36. Duplicate functions in Validator contract

Risk Level: Info

Status: Fixed

Contracts:

- contracts/Validator.sol

Location: Function: `claimReward()` `claimRewardFor()`.

Description:

The Validator contract contains two identical functions: `claimReward()` and `claimRewardFor()`. Both functions perform exactly the same operations with the same logic, making one of them redundant. This duplication increases contract size and gas costs unnecessarily.

Remediation:

Remove the duplicate function `claimRewardFor()` and keep only `claimReward()`.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.