

NodeSet Constellation Smart Contracts Review | September 2024

Table of contents

- 1. Introduction
 - Defining Severity
 - Referencing updated code
 - Disclaimer
- 2. Executive Summary
- 3. Critical Bugs and Vulnerabilities
- 4. Line-by-line review
 - contracts/Constellation/Utils/Directory.sol
 - o contracts/Constellation/Utils/UpgradeableBase.sol
 - contracts/Constellation/MerkleClaimStreamer.sol
 - contracts/Constellation/OperatorDistributor.sol
 - contracts/Constellation/PoAConstellationOracle.sol
 - contracts/Constellation/RPLVault.sol
 - contracts/Constellation/SuperNodeAccount.sol
 - contracts/Constellation/WETHVault.sol
 - contracts/Constellation/Whitelist.sol
 - contracts/External/NodeSetOperatorRewardDistributor.sol
 - contracts/External/Treasury.sol

1. Introduction

Date	Auditor(s)
September 2024	Oleksii Matiiasevych, Anderson Lee

NodeSet requested **ChainSafe Systems** to perform a review of the constellation smart contracts. The contracts can be identified by the following git commit hash: d97f0e5867af5ecca1933b081a1ae074b73cf587

There are 15 contracts in scope.

After the initial review, **NodeSet** team applied a number of updates which can be identified by the following git commit hash: e1046be648879972669ea5c8177278f21043b487

Additional verification was performed after that.

Defining Severity

Each finding is assigned a severity level.

Note	Notes are informational in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
Optimization	Optimizations are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Minor	Minor issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Major	Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
Critical	Critical issues are directly exploitable security vulnerabilities that need to be fixed.

Referencing updated code

Resolved	The finding has been acknowledged and the team has since updated the code.
Rejected	The team dismissed this finding and no changes will be made.

Disclaimer

The review makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts for any specific purpose, or their bug free status.

2. Executive Summary

All the minor and higher severity issues were fixed and are not present in the final version of the contract.

There are **no** known compiler bugs for the specified compiler version (0.8.17), that might affect the contracts' logic.

There were **6 majors**, **10 minors**, 75 informational/optimization issues identified in the initial version of the contracts. The major/minor issues found in the contracts were not present in the final version of the contracts. They are described below for historical purposes. During the audit, the NodeSet team identified and fixed some of the major/minor issues on their own.

A compromise of the **ADMIN_ORACLE** signer key in the **PoAConstellationOracle** could result in the available liquidity being drained from the protocol. NodeSet team is aware of this weakness and plans to strengthen it in the future upgrades, while relying on the the operational security at the beginning.

We are looking forward to future engagements with the NodeSet team.

3. Critical Bugs and Vulnerabilities

No critical issues were identified.

4. Line-by-line review

contracts/Constellation/Utils/Directory.sol

L32 Note Resolved

The Directory contract does not have a getter for Protocol.sanctions address.

L257 Note Resolved

The initialize() function does not validate merkleClaimStreamer address value. Even if it is 0x0 it will get a core protocol role assigned.

L260 Note Resolved

The initialize() function does not validate superNode address value. Even if it is 0x0 it will get a core protocol role assigned.

L358 Note Resolved

The setTreasurer() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L364 Note Resolved

The disableSactions() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L370 Note Resolved

The enableSactions() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L377 Note Resolved

The enableOracle() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L384 Minor Resolved

require(hasRole(Constants.TIMELOCK_SHORT, msg.sender),
Constants.ADMIN_ONLY_ERROR);

The setAll() function access error message is misleading. It is only admin, while it should be only timelock short.

L385 Note Resolved

The setAll() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

contracts/Constellation/Utils/UpgradeableBase.sol



The initialize() function does not initialize the ReentrancyGuard parent contract. First execution of a nonReentrant modifier will cost extra gas.



L100

Major

```
function initialize(address directoryAddress) public virtual initializer
{
    _directory = Directory(directoryAddress);
    __UUPSUpgradeable_init();
}
```

The initialize() function of the base/parent contracts is meant to have an onlyInitializing() modifier instead of initializer(), to allow initialization in a call separate from deployment.

contracts/Constellation/MerkleClaimStreamer.sol

hardhat/console.sol import could be removed in the contract.

Resolved

```
function sweepLockedTVL() public onlyProtocolOrAdmin {
     ...
     if(priorRplStreamAmount > 0){
          SafeERC20.safeTransfer(IERC20(_directory.getRPLAddress()),
          getDirectory().getMerkleClaimStreamerAddress(), priorRplStreamAmount);
          ...
     }
}
```

The sweepLockedTVL() function transfers the priorRplStreamAmount to itself instead of the OperatorDistributor.

contracts/Constellation/OperatorDistributor.sol

L54 Note Resolved

The setTargetStakeRatio() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L66 Note Resolved

The setMinimumStakeRatio() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L128 Major Resolved

The calculateRplStakeShortfall() function incorrectly calculates matchedStakeRatio. The correct formula would be matchedStakeRatio = _existingRplStake.mulDiv(1e36, _rpEthMatched * ethPriceInRpl).

L205 Major Resolved

```
function rebalanceRplStake(uint256 _ethStaked) public {
    ...
}
```

The rebalanceRplStake() function doesn't have access control and can be called by anyone passing the manipulated _ethStaked value.

L219 Optimization Resolved

The rebalanceRplStake() function has an always false condition stakeIncrease == 0.



The processMinipool() function could be called by anyone to process minipools out of any particular order.

L394 Optimization Resolved

The rebalanceWethVault() does an excessive weth.withdraw() in case there was a weth.deposit() needed.

L399 Minor Resolved

```
function rebalanceWethVault() public onlyProtocol {
    ...
    if (balanceEthAndWeth >= requiredWeth) {
          ...
    } else {
          ...
        weth.deposit{value: address(this).balance}();
          SafeERC20.safeTransfer(IERC20(address(weth)), address(vweth),
address(this).balance);
    }
}
```

The rebalanceWethVault() function sends 0 WETH to the vweth in case it has less than required.

L405 Note Rejected

The rebalanceRplVault() function only does a one way rebalancing towards replenishing the vault, but not taking back if there is too much there.

L419 Optimization Resolved

The rebalanceRplVault() function will make a zero transfer in case there is enough RPL in the vault.

L448 Optimization Resolved

The onEthBeaconRewardsReceived() function calls the onIncreaseOracleError() function on this same contract, but tries to get it address from the directory instead.

L486 Optimization Resolved

The provisionLiquiditiesForMinipoolCreation() function calls _directory.getSuperNodeAddress() twice.

contracts/Constellation/PoAConstellationOracle.sol

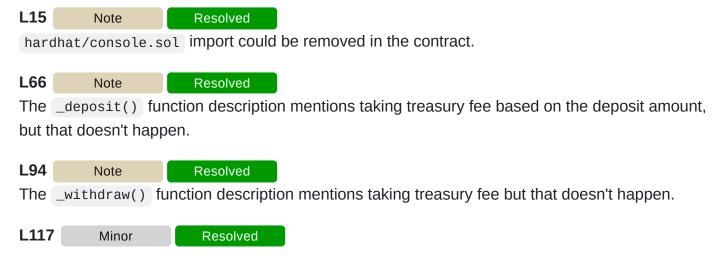


The oracle signer's private key could be used to publish manipulated yield value which could be used to drain all available WETH liquidity from the WETHVault.



The setTotalYieldAccrued() function could be executed multiple times with the same oracle signature as long as the sigData.timeStamp is in the future. Consider making sure that the sigData.timeStamp has already passed.

contracts/Constellation/RPLVault.sol



```
function _withdraw(
   address caller,
   address receiver,
   address owner,
   uint256 assets,
   uint256 shares
) internal virtual override {
    ...
   od.processNextMinipool();
   super._withdraw(caller, receiver, owner, assets, shares);
```

} ...

The _withdraw() function processes a minipool before the actual withdrawal with an intention to make it more likely to succeed, but there is already enough liquidity due to the requirement above. Besides, the minipool processing does not involve moving the liquidity to the RPLVault.

L168 Note Resolved

The setTreasuryFee() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L179 Note Resolved

The setMinWethRplRatio() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

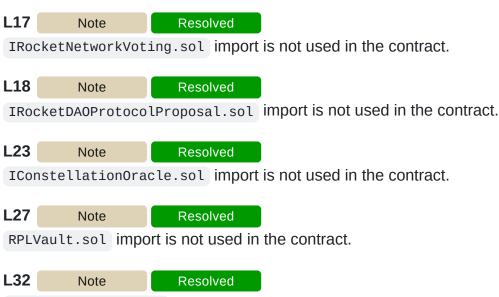
L192 Optimization Resolved

The setLiquidityReservePercent() function has an always true requirement that checks that the new percent is >= 0.

L201 Note Resolved

The setLiquidityReservePercent() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

contracts/Constellation/SuperNodeAccount.sol



hardhat/console.sol import could be removed in the contract.

L54 Note Resolved

The adminServerSigExpiry storage value is not used in the signatures verification.



The createMinipool() function description has outdated list of properties for the CreateMinipoolConfig struct.

L212 Minor Resolved

```
function createMinipool(CreateMinipoolConfig calldata _config) public
payable {
    ...
    require(hasSufficientLiquidity(bond), 'NodeAccount: protocol must
have enough rpl and eth');
    ...

OperatorDistributor(_directory.getOperatorDistributorAddress()).provision
LiquiditiesForMinipoolCreation(bond);
    ...
}
```

The createMinipool() function calls provisionLiquiditiesForMinipoolCreation() after checking hasSufficientLiquidity(bond) which could result in the situation where liquidity becomes insufficient after the provision, because provision rebalances vaults.

L217 Note Resolved

The createMinipool() function excessively relies on the uniquness of signatures. Due to an ECDSA signatures malleability user could craft a second valid signature for each message. The message itself is already unique due to the nonces[subNodeOperator] present in it, which is by itself a sufficient measure to restrict signatures reuse.

L253 Optimization Resolved

The createMinipool() function calls _directory.getOperatorDistributorAddress() function twice.

L258 Optimization Resolved

The createMinipool() function makes an excessive external call to itself to execute this.getEthStaked(). Consider removing this keyword to execute the function internally.

L319 Optimization Resolved

The stake() function refund block condition is always true. Consider making it lockupBalance > 0.

L337 Note Resolved

The closeDissolvedMinipool() has an excessive subNodeOperatorAddress parameter which is fetched from the minipoolData anyway.

L458 Optimization Resolved

The hasSufficientLiquidity() function makes an excessive external call to this.getEthMatched() function.

contracts/Constellation/WETHVault.sol





The _withdraw() function processes a minipool before the actual withdrawal with an intention to make it more likely to succeed, but there is already enough liquidity due to the requirement above. Perhaps the reugirement should be moved to after the minipool processing.

```
function previewMint(uint256 shares) public view virtual override returns (uint256) {
```

```
uint256 assets = super.previewMint(shares);
return assets + this.getMintFeePortion(assets);
}
```

The previewMint() function incorrectly calculates the assets needed, resulting in that WETHVault.mint() minting more shares than WETHVault.deposit() for the same amount of assets.

L185 Note Rejected

The totalAssets() function could potentially underflow and report a very high and incorrect number. Consider removing int -> uint casting.

L222 Note Resolved

The getMissingLiquidityAfterDepositNoFee() function is marked as dev/testing but is still used in the live code path.

L276 Note Resolved

The setMaxWethRplRatio() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L286 Note Resolved

The comparison between _treasuryFee and le18 is not needed because the sum of _treasuryFee and nodeOperatorFee is compared with le18 in the next line.

L288 Note Resolved

The setTreasuryFee() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L298 Note Resolved

The comparison between __nodeOperatorFee and _le18 is not needed because the sum of treasuryFee and __nodeOperatorFee is compared with _le18 in the next line.

L300 Note Resolved

The setNodeOperatorFee() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L311 Note Resolved

The comparison between _treasuryFee and le18 is not needed because the sum of _treasuryFee and _nodeOperatorFee is compared with le18 later.

L312 Note Resolved

The comparison between __nodeOperatorFee and le18 is not needed because the sum of __treasuryFee and __nodeOperatorFee is compared with le18 in the next line.



The setProtocolFees() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

L328 Optimization Resolved

The setLiquidityReservePercent() function has an always true requirement that checks that the new percent is >= 0.

L340 Optimization Resolved

The setMintFee() function has an always true requirement part that checks that the newMintFee is >= 0.



The setMintFee() function changes the contract storage but doesn't emit an event which makes it harder to monitor the system.

contracts/Constellation/Whitelist.sol



The Operator struct fields could be shrunk to fit a single storage slot.

L39 Note Resolved

The whitelistSigExpiry storage variable is not used in signatures verification.

L90 Note Resolved

The getOperatorIndex() function is private and is not used in the contract.

L92 Minor Resolved

```
function getOperatorIndex(address a) private view returns (uint) {
    ...
    return reverseNodeIndexMap[a];
}
```

The <code>getOperatorIndex()</code> function returns incorrect results. Reverse index is kept as <code>+1</code> from the actual index, so the result should be subtracted.

```
L162 Major Resolved
```

```
function _removeOperator(address nodeOperator) internal {
    ...
```

```
numOperators--;
}
```

The _removeOperator() function decreases the numOperators which is used as an index for new operators addition. This will introduce storage corruption as multiple operators could get the same index assigned. One way to remedy this is to use EnumerableSet.AddressSet library to manage operators list instead.

L181 Major Resolved

```
function addOperators(address[] memory operators, bytes[] memory _sig)
public {
    for (uint i = 0; i < operators.length; i++) {
        require(!_permissions[operators[i]],
    Constants.OPERATOR_DUPLICATE_ERROR);
    }
    for (uint i = 0; i < operators.length; i++) {
        _addOperator(operators[i], _sig[i]);
    }
    emit OperatorsAdded(operators);
}</pre>
```

The addoperators() function verifies duplication of all new operators before adding them. If the input operators[] array has duplicates, then they will pass the verification and introduce storage corruption. One way to remedy this is to verify duplication sequentially together with _addoperator() function execution.

L217 Note Resolved

The _addoperator() function excessively relies on the uniquness of signatures. Due to an ECDSA signatures malleability user could craft a second valid signature for each message. The message itself is already unique due to the <code>nonces[_operator]</code> present in it, which is by itself a sufficient measure to restrict signatures reuse.

L233 Optimization Resolved

The _addoperator() function reads numoperators twice from storage. Consider caching it in a local variable instead.

contracts/External/NodeSetOperatorRewardDistributor.sol

L45 Note Rejected

The initialize() function does not initialize the ReentrancyGuard parent contract. First execution of a nonReentrant modifier will cost extra gas.

L46 Note Resolved

The initialize() function grants the nodeset admin role which cannot be revoked or transferred at a later date without an upgrade, because no one is holding the admin role.

L52 Note Resolved

The initialize() function excessively revokes admin role from msg.sender, even though msg.sender doesn't have it.

L97 Optimization Resolved

The claimRewards() function makes an excessive external call to the RewardsDistributor contract itself for signer verification by specifying this in front of the hasRole() function invocation. Consider removing this to save gas.

L109 Minor Resolved



The claimRewards() function increments the nonces[_did] value after making an unsafe external call, which would result signatures reuse if there were no reentrancy protection. It is recommended to update the state of the contract before making any external calls to avoid such pitfalls. Such modification will also make the reentrancy protection unnecessary.

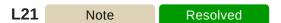
L116 Optimization Resolved

The invalidateAllOutstandingSigs() function makes an excessive external call to the RewardsDistributor contract itself for access verification by specifying this in front of the hasRole() function invocation. Consider removing this to save gas.

L124 Optimization Resolved

The invalidateAllOutstandingSig() function makes an excessive external call to the RewardsDistributor contract itself for access verification by specifying this in front of the hasRole() function invocation. Consider removing this to save gas.

contracts/External/Treasury.sol



The BAD_TREASURY_EXECUTION_ERROR constant is not used.

L34 Note Rejected

The initialize() function does not initialize the ReentrancyGuard parent contract. First execution of a nonReentrant modifier will cost extra gas.

L36 Note Resolved

The initialize() function grants the treasurer role which cannot be revoked or transferred at a later date without an upgrade, because no one is holding the admin role.

L37 Note Resolved

The initialize() function excessively revokes admin role from msg.sender, even though msg.sender doesn't have it.

L48 Minor Resolved

function _claimTokenInternal(address _tokenAddress, address _to, uint256
_amount) internal {
 IERC20(_tokenAddress).transfer(_to, _amount);
 ...
}

The _claimTokenInternal() function would revert if trying to transfer a ERC20 token that does not return bool on transfers. Consider using a SafeERC20 library to support more tokens.

L53 Note Resolved

The _claimEthInternal() function uses a address.transfer() function to transfer ETH. It is recommended to use address.call() instead to make sure it won't fail with out of gas error.

L69 Optimization Resolved

The onlyTreasurer() modifier makes an excessive external call to the Treasury contract itself for access verification by specifying this in front of the hasRole() function invocation. Consider removing this to save gas.

L113 Optimization Resolved

The executeAll() function excessively reads _targets.length value from calldata on each loop iteration. Consider caching it in a local variable instead.