Symbiotic Relay Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

0kage

Farouk

ChainDefenders (@1337web3 & @PeterSRWeb3)

Assisting Auditors

Aleph-v (Cryptography)

September 11, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Symbiotic SDK is a comprehensive middleware infrastructure that enables permissionless validator management and signature aggregation for decentralized networks. System is designed to collect and aggregate signatures from validators, maintain validator sets, and facilitate settlement of network state.

5 Audit Scope

The Symbiotic Middleware SDK is a comprehensive framework that enables the development of custom middleware solutions for restaking protocols. The scope excludes Symbiotic Core contracts and focuses exclusively on the middleware SDK components that enable custom middleware development and deployment within the Symbiotic ecosystem.

All files in the `src/**` folders were part of the audit scope except for the `BN254.sol`.

The scope of this audit was limited to:

```
relay-contracts/src/contracts/libraries/keys/KeyBlsBn254.sol
relay-contracts/src/contracts/libraries/keys/KeyEcdsaSecp256k1.sol
relay-contracts/src/contracts/libraries/sigs/SigBlsBn254.sol
relay-contracts/src/contracts/libraries/sigs/SigEcdsaSecp256k1.sol
relay-contracts/src/contracts/libraries/structs/Checkpoints.sol
relay-contracts/src/contracts/libraries/structs/PersistentSet.sol
relay-contracts/src/contracts/libraries/utils/InputNormalizer.sol
relay-contracts/src/contracts/libraries/utils/KeyTags.sol
relay-contracts/src/contracts/libraries/utils/Scaler.sol
relay-contracts/src/contracts/libraries/utils/ValSetVerifier.sol
relay-contracts/src/contracts/modules/base/NetworkManager.sol
relay-contracts/src/contracts/modules/base/OzEIP712.sol
relay-contracts/src/contracts/modules/base/PermissionManager.sol
relay-contracts/src/contracts/modules/common/permissions/OzAccessControl.sol
relay-contracts/src/contracts/modules/common/permissions/OzAccessManaged.sol
```

```
relay-contracts/src/contracts/modules/common/permissions/OzOwnable.sol
relay-contracts/src/contracts/modules/key-registry/KeyRegistry.sol
relay-contracts/src/contracts/modules/network/Network.sol
relay-contracts/src/contracts/modules/settlement/sig-verifiers/libraries/ExtraDataStorageHelper.sol
relay-contracts/src/contracts/modules/settlement/sig-verifiers/SigVerifierBlsBn254Simple.sol
relay-contracts/src/contracts/modules/settlement/sig-verifiers/SigVerifierBlsBn254ZK.sol
relay-contracts/src/contracts/modules/settlement/Settlement.sol
relay-contracts/src/contracts/modules/valset-driver/EpochManager.sol
relay-contracts/src/contracts/modules/valset-driver/ValSetDriver.sol
relay-contracts/src/contracts/modules/voting-power/base/VotingPowerCalcManager.sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/libraries/ChainlinkPriceFee
↪  d.sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/EqualStakeVPCalc.sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/NormalizedTokenDecimalsVPCa
↪  lc.sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/PricedTokensChainlinkVPCalc
↪  .sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/WeightedTokensVPCalc.sol
relay-contracts/src/contracts/modules/voting-power/common/voting-power-calc/WeightedVaultsVPCalc.sol
relay-contracts/src/contracts/modules/voting-power/extensions/logic/BaseRewardsLogic.sol
relay-contracts/src/contracts/modules/voting-power/extensions/logic/BaseSlashingLogic.sol
relay-contracts/src/contracts/modules/voting-power/extensions/logic/OpNetVaultAutoDeployLogic.sol
relay-contracts/src/contracts/modules/voting-power/extensions/BaseRewards.sol
relay-contracts/src/contracts/modules/voting-power/extensions/BaseSlashing.sol
relay-contracts/src/contracts/modules/voting-power/extensions/MultiToken.sol
relay-contracts/src/contracts/modules/voting-power/extensions/OperatorsBlacklist.sol
relay-contracts/src/contracts/modules/voting-power/extensions/OperatorsJail.sol
relay-contracts/src/contracts/modules/voting-power/extensions/OperatorsWhitelist.sol
relay-contracts/src/contracts/modules/voting-power/extensions/OperatorVaults.sol
relay-contracts/src/contracts/modules/voting-power/extensions/OpNetVaultAutoDeploy.sol
relay-contracts/src/contracts/modules/voting-power/extensions/SharedVaults.sol
relay-contracts/src/contracts/modules/voting-power/logic/VotingPowerProviderLogic.sol
relay-contracts/src/contracts/modules/voting-power/VotingPowerProvider.sol
```

6 Executive Summary

Over the course of 15 days, the Cyfrin team conducted an audit on the Symbiotic Relay smart contracts provided by Symbiotic. In this period, a total of 13 issues were found.

The Symbiotic Middleware SDK is a comprehensive framework that enables the development of custom middleware solutions for restaking protocols. The SDK provides modular components for operator registration, stake management, signature verification, voting power calculation, and network governance. Key components include settlement modules with various signature verification schemes (BLS BN254, ECDSA), voting power providers with support for weighted calculations, and network management capabilities with timelock functionality for secure operations.

The comprehensive test suite demonstrates excellent code quality and defensive programming practices throughout the codebase. The protocol implements robust access controls, comprehensive input validation, and well-structured modular architecture that minimizes attack surfaces.

Summary

Project Name	Symbiotic Relay
Repository	relay-contracts
Commit	1a1f94fa1798...
Fix Commit	054c0c5d6d1f...
Audit Timeline	July 15th - August 4th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	2
Informational	5
Gas Optimizations	3
Total Issues	13

Summary of Findings

[M-1] Voting power returns 0 during oracle failures causing unfair voting outcomes	Acknowledged
[M-2] Precision loss causes systematic under-reporting of voting power for high decimal tokens	Acknowledged
[M-3] Incomplete validation in `OpNetVaultAutoDeploy` allows invalid burner configuration for registering operators with auto vault deployments	Resolved
[L-1] Missing bounds check on weight values in `WeightedTokensVPCalc` and `WeightedVaultsVPCalc`	Resolved
[L-2] `unwhitelistOperator` allows state changes when whitelist Is disabled, causing inconsistent operator state	Resolved
[I-1] `OpNetVaultAutoDeployLogic::getVetoSlasherParams` is never used	Acknowledged
[I-2] `__NoncesUpgradeable_init` is not invoked	Acknowledged
[I-3] Redundant manual access control checks	Acknowledged
[I-4] Unused functions in `KeyRegistry`	Acknowledged
[I-5] Inconsistent error handling for empty data	Resolved
[G-1] `PersistentSet` library storage access	Resolved
[G-2] Cache storage reads in `upperLookupRecentCheckpoint`	Acknowledged
[G-3] Loop can use unchecked arithmetic and cache array length	Acknowledged

7 Findings

7.1 Medium Risk

7.1.1 Voting power returns 0 during oracle failures causing unfair voting outcomes

Description: The `PricedTokensChainlinkVPCalc::stakeToVotingPowerAt` silently returns zero voting power when Chainlink oracles fail or return stale data. Operators lose all voting power without notification creating unintended voting outcomes.

The core issue lies in the `ChainlinkPriceFeed.getPriceAt()` function's error handling:

```
// ChainlinkPriceFeed.sol
function getPriceAt(address aggregator, uint48 timestamp, bool invert, uint48 stalenessDuration)
    public view returns (uint256) {
    (bool success, RoundData memory roundData) = getPriceDataAt(aggregator, timestamp, invert,
    ↪  stalenessDuration);
    return success ? roundData.answer : 0; // @audit silently returns 0
}
```

This zero value propagates through the voting power calculation chain:

```
// PricedTokensChainlinkVPCalc.sol
function stakeToVotingPowerAt(address vault, uint256 stake, bytes memory extraData, uint48 timestamp)
    public view virtual override returns (uint256) {
    return super.stakeToVotingPowerAt(vault, stake, extraData, timestamp) *
    ↪  getTokenPriceAt(_getCollateral(vault), timestamp); // @audit returns 0 when oracle is stale
}
```

Impact: If oracles return stale data at the voting timestamp, operator can lose all voting power even though he has active stake.

Proof of Concept: Add the following PoC in test->modules->common->voting-power-calc

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.25;

import "forge-std/Test.sol";

import {VotingPowerProvider} from
↪  "../../../../../src/contracts/modules/voting-power/VotingPowerProvider.sol";
import {VotingPowerProviderLogic} from
    "../../../../../src/contracts/modules/voting-power/logic/VotingPowerProviderLogic.sol";
import {MultiToken} from "../../../../../src/contracts/modules/voting-power/extensions/MultiToken.sol";
import {IVotingPowerProvider} from
↪  "../../../../../src/interfaces/modules/voting-power/IVotingPowerProvider.sol";
import {INetworkManager} from "../../../../../src/interfaces/modules/base/INetworkManager.sol";
import {IOzEIP712} from "../../../../../src/interfaces/modules/base/IOzEIP712.sol";
import {NoPermissionManager} from "../../../../../test/mocks/NoPermissionManager.sol";
import {PricedTokensChainlinkVPCalc} from
    "../../../../../src/contracts/modules/voting-power/common/voting-power-calc/PricedTokensChainlinkVP
    ↪  Calc.sol";
import {OperatorVaults} from
↪  "../../../../../src/contracts/modules/voting-power/extensions/OperatorVaults.sol";
import {ChainlinkPriceFeed} from
    "../../../../../src/contracts/modules/voting-power/common/voting-power-calc/libraries/ChainlinkPric
    ↪  eFeed.sol";

import {BN254} from "../../../../../src/contracts/libraries/utils/BN254.sol";
import {AggregatorV3Interface} from
↪  "@chainlink/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol";
import "../../../../InitSetup.sol";
```

```solidity
// Mock Chainlink Aggregator that simulates stale data conditions
contract MockChainlinkAggregator is AggregatorV3Interface {
    uint80 private _roundId;
    int256 private _answer;
    uint256 private _startedAt;
    uint256 private _updatedAt;
    uint80 private _answeredInRound;
    uint8 private _decimals;

    bool public isStale;
    bool public shouldRevert;

    constructor(uint8 decimals_) {
        _decimals = decimals_;
        _roundId = 1;
        _answer = 2000e8; // $2000 price
        _startedAt = block.timestamp;
        _updatedAt = block.timestamp;
        _answeredInRound = 1;
        isStale = false;
        shouldRevert = false;
    }

    function decimals() external view override returns (uint8) {
        return _decimals;
    }

    function description() external pure override returns (string memory) {
        return "Mock Chainlink Aggregator";
    }

    function version() external pure override returns (uint256) {
        return 1;
    }

    function getRoundData(uint80 roundId) external view override
        returns (uint80, int256, uint256, uint256, uint80) {
        if (shouldRevert) {
            revert("Oracle failure");
        }
        return (_roundId, _answer, _startedAt, _updatedAt, _answeredInRound);
    }

    function latestRoundData() external view override
        returns (uint80, int256, uint256, uint256, uint80) {
        if (shouldRevert) {
            revert("Oracle failure");
        }

        uint256 updatedAt = _updatedAt;
        if (isStale) {
            // Make data appear stale by setting updatedAt to a very old timestamp
            updatedAt = block.timestamp - 1 days;
        }

        return (_roundId, _answer, _startedAt, updatedAt, _answeredInRound);
    }

    // Test helpers to simulate oracle conditions
    function setStale(bool _isStale) external {
        isStale = _isStale;
    }
```

```
    function setShouldRevert(bool _shouldRevert) external {
        shouldRevert = _shouldRevert;
    }

    function setPrice(int256 newPrice) external {
        _answer = newPrice;
        _roundId++;
        _updatedAt = block.timestamp;
        _answeredInRound = _roundId;
    }

    function setUpdatedAt(uint256 newUpdatedAt) external {
        _updatedAt = newUpdatedAt;
    }
}

// Test voting power provider that uses PricedTokensChainlinkVPCalc
contract TestVotingPowerProvider is VotingPowerProvider, PricedTokensChainlinkVPCalc,
↪  NoPermissionManager {
    constructor(address operatorRegistry, address vaultFactory) VotingPowerProvider(operatorRegistry,
    ↪  vaultFactory) {}

    function initialize(
        IVotingPowerProvider.VotingPowerProviderInitParams memory votingPowerProviderInit
    ) external initializer {
        __VotingPowerProvider_init(votingPowerProviderInit);
    }

    function registerOperator(address operator) external {
        _registerOperator(operator);
    }

    function unregisterOperator(address operator) external {
        _unregisterOperator(operator);
    }

    function setSlashingWindow(uint48 sw) external {
        _setSlashingWindow(sw);
    }

    function registerToken(address token) external {
        _registerToken(token);
    }

    function unregisterToken(address token) external {
        _unregisterToken(token);
    }

    function registerSharedVault(address vault) external {
        _registerSharedVault(vault);
    }

    function unregisterSharedVault(address vault) external {
        _unregisterSharedVault(vault);
    }

    function registerOperatorVault(address operator, address vault) external {
        _registerOperatorVault(operator, vault);
    }

    function unregisterOperatorVault(address operator, address vault) external {
        _unregisterOperatorVault(operator, vault);
    }
```

```
    // Expose the setTokenHops function for testing
    function setTokenHops(
        address token,
        address[2] memory aggregators,
        bool[2] memory inverts,
        uint48[2] memory stalenessDurations
    ) public override {
        _setTokenHops(token, aggregators, inverts, stalenessDurations);
    }
}

contract PricedTokensChainlinkVPCalcTest is InitSetupTest {
    TestVotingPowerProvider private votingPowerProvider;
    MockChainlinkAggregator private mockAggregator;
    address private vault;

    address operator1 = address(0xAAA1);
    address operator2 = address(0xAAA2);

    uint256 private constant STAKE_AMOUNT = 1000e18; // 1000 tokens
    uint48 private constant STALENESS_DURATION = 1 hours;

     function setUp() public override {
        InitSetupTest.setUp();

        address token = initSetupParams.masterChain.tokens[0];
        vault = initSetupParams.masterChain.vaults[0];

        // Deploy mock aggregator (8 decimals like most Chainlink feeds)
        mockAggregator = new MockChainlinkAggregator(8);

        // Create a mock vault for testing
        vault = makeAddr("vault");

        // Deploy voting power provider
        votingPowerProvider = new TestVotingPowerProvider(
            address(symbioticCore.operatorRegistry),
            address(symbioticCore.vaultFactory)
        );

          // Initialize the voting power provider
        INetworkManager.NetworkManagerInitParams memory netInit =
            INetworkManager.NetworkManagerInitParams({network: vars.network.addr, subnetworkID:
            ↪  IDENTIFIER});
        IVotingPowerProvider.VotingPowerProviderInitParams memory initParams = IVotingPowerProvider
            .VotingPowerProviderInitParams({
                networkManagerInitParams: netInit,
                ozEip712InitParams: IOzEIP712.OzEIP712InitParams({name: "MyVotingPowerProvider",
                ↪  version: "1"}),
                slashingWindow: 100,
                token: token
        });

        votingPowerProvider.initialize(initParams);
        _networkSetMiddleware_SymbioticCore(vars.network.addr, address(votingPowerProvider));


        _registerOperator_SymbioticCore(symbioticCore, operator1);
        _registerOperator_SymbioticCore(symbioticCore, operator2);

         vm.startPrank(operator1);
        votingPowerProvider.registerOperator(operator1);
```

```
        vm.stopPrank();

        vm.startPrank(operator2);
        votingPowerProvider.registerOperator(operator2);
        vm.stopPrank();

          // Create a proper vault for operator1 with correct parameters
        vault = _getVault_SymbioticCore(
            VaultParams({
                owner: operator1,
                collateral: token,
                burner: 0x000000000000000000000000000000000000dEaD,
                epochDuration: votingPowerProvider.getSlashingWindow() * 2,
                whitelistedDepositors: new address[](0),
                depositLimit: 0,
                delegatorIndex: 2, // OPERATOR_SPECIFIC delegator
                hook: address(0),
                network: address(0),
                withSlasher: true,
                slasherIndex: 0,
                vetoDuration: 1
            })
        );

         _operatorOptIn_SymbioticCore(operator1, vault);

          _networkSetMaxNetworkLimit_SymbioticCore(
            votingPowerProvider.NETWORK(),
            vault,
            votingPowerProvider.SUBNETWORK_IDENTIFIER(),
            type(uint256).max
        );

        _curatorSetNetworkLimit_SymbioticCore(
            operator1, vault, votingPowerProvider.SUBNETWORK(), type(uint256).max
        );

        _deal_Symbiotic(token, getStaker(0).addr, STAKE_AMOUNT *10, true); // some big number
        _stakerDeposit_SymbioticCore(getStaker(0).addr, vault, STAKE_AMOUNT);

        vm.startPrank(vars.network.addr);
        votingPowerProvider.registerOperatorVault(operator1, vault);
        vm.stopPrank();

          // Set up token price hops with our mock aggregator
        address[2] memory aggregators = [address(mockAggregator), address(0)];
        bool[2] memory inverts = [false, false];
        uint48[2] memory stalenessDurations = [STALENESS_DURATION, 0];

        votingPowerProvider.setTokenHops(token, aggregators, inverts, stalenessDurations);

    }

    function test_NormalPriceReturnsCorrectVotingPower() public {
        // Set oracle to return price = 2000 (2000e8 in 8 decimals)
        mockAggregator.setPrice(2000e8);

        //Calculate voting power
        uint256 votingPower = votingPowerProvider.stakeToVotingPower(vault, STAKE_AMOUNT, "");

        assertEq(votingPower, (STAKE_AMOUNT * 10**(24-18)) * 2000*10**8*10**(18-8), "Voting power should
        ↪  be stake multiplied by price"); // normalization scaling to 10^24
```

```
        // 2000*10**8*10**(18-8) -> chainlink price is scaled to 18 decimals
        // voting power is normalized to 24 decimals
    }

    function test_StaleOracleReturnsZeroVotingPower() public {
        // Set oracle to stale
        mockAggregator.setStale(true);

        // Calculate voting power
        uint256 votingPower = votingPowerProvider.stakeToVotingPower(vault, STAKE_AMOUNT, "");

        // Voting power is 0 due to stale data - silently distorts the voting outcomes
        assertEq(votingPower, 0, "Voting power should be zero when oracle data is stale");
    }

}
```

Recommended Mitigation: Consider returning last known good price instead of returning 0. The voting power inaccuracy attributed to last known good price is still much better than making voting power 0

Symbiotic: Acknowledged.

Cyfrin: Acknowledged.

7.1.2 Precision loss causes systematic under-reporting of voting power for high decimal tokens

Description: The `NormalizedTokenDecimalsVPCalc::_normalizeVaultTokenDecimals()` causes voting power loss for legitimate stakeholders when high-decimal tokens (>24 decimals) are used. This occurs due to integer division in the Scaler.scale() function, which rounds down small results to zero.

```
// NormalizedTokenDecimalsVPCalc.sol
function _normalizeVaultTokenDecimals(address vault, uint256 votingPower) internal view virtual returns
↪  (uint256) {
    return votingPower.scale(IERC20Metadata(_getCollateral(vault)).decimals(), BASE_DECIMALS); //@audit
    ↪  BASE_DECIMALS = 24
}

//Scaler.sol
function scale(uint256 value, uint8 decimals, uint8 targetDecimals) internal pure returns (uint256) {
    if (decimals > targetDecimals) {
        uint256 decimalsDiff;
        unchecked {
            decimalsDiff = decimals - targetDecimals;
        }
        return value / 10 ** decimalsDiff;  // @audit precision loss
    }
    // ...
}
```

When decimals > BASE_DECIMALS (24), the function divides by 10^(decimals-24), causing stakes smaller than this divisor to become zero voting power.

```
// Divisor = 10^(30-24) = 10^6 = 1,000,000
Stake: 999,999 units → Voting Power: 0
Stake: 500,000 units → Voting Power: 0
Stake: 1,000,000 units → Voting Power: 1
```

All voting power calculators that extend `NormalizedTokenDecimalsVPCalc` are affected.

Impact: Legitimate stakeholders with meaningful economic stakes can lose voting power.

Proof of Concept: Add this to `NormalizedTokenDecimalsVPCalc.t.sol`

```
function test_PrecisionLossPoC_30Decimals() public {
    // Even worse case with 30 decimals
    votingPowerProvider =
        new TestVotingPowerProvider(address(symbioticCore.operatorRegistry),
        ↪  address(symbioticCore.vaultFactory));

    INetworkManager.NetworkManagerInitParams memory netInit =
        INetworkManager.NetworkManagerInitParams({network: vars.network.addr, subnetworkID:
        ↪  IDENTIFIER});

    // Create token with 30 decimals (6 more than BASE_DECIMALS=24)
    MockToken mockToken = new MockToken("MockToken", "MTK", 30);

    IVotingPowerProvider.VotingPowerProviderInitParams memory votingPowerProviderInit =
    ↪  IVotingPowerProvider
        .VotingPowerProviderInitParams({
        networkManagerInitParams: netInit,
        ozEip712InitParams: IOzEIP712.OzEIP712InitParams({name: "MyVotingPowerProvider", version: "1"}),
        slashingWindow: 100,
        token: address(mockToken)
    });

    votingPowerProvider.initialize(votingPowerProviderInit);
    _networkSetMiddleware_SymbioticCore(vars.network.addr, address(votingPowerProvider));

    // Demonstrate extreme precision loss
    // Divisor = 10^(30-24) = 10^6 = 1,000,000

    Vm.Wallet memory operator = getOperator(0);
    vm.startPrank(operator.addr);
    votingPowerProvider.registerOperator(operator.addr);
    vm.stopPrank();

    address operatorVault = _getVault_SymbioticCore(
        VaultParams({
            owner: operator.addr,
            collateral: address(mockToken),
            burner: 0x000000000000000000000000000000000000dEaD,
            epochDuration: votingPowerProvider.getSlashingWindow() * 2,
            whitelistedDepositors: new address[](0),
            depositLimit: 0,
            delegatorIndex: 2,
            hook: address(0),
            network: address(0),
            withSlasher: true,
            slasherIndex: 0,
            vetoDuration: 1
        })
    );

    _operatorOptIn_SymbioticCore(operator.addr, operatorVault);
    _networkSetMaxNetworkLimit_SymbioticCore(
        votingPowerProvider.NETWORK(),
        operatorVault,
        votingPowerProvider.SUBNETWORK_IDENTIFIER(),
        type(uint256).max
    );
    _curatorSetNetworkLimit_SymbioticCore(
        operator.addr, operatorVault, votingPowerProvider.SUBNETWORK(), type(uint256).max
    );

    _deal_Symbiotic(address(mockToken), getStaker(0).addr, type(uint128).max, true);
```

```
    // Deposit 999,999 units - substantial economic value but < 1,000,000 divisor
    _stakerDeposit_SymbioticCore(getStaker(0).addr, operatorVault, 999_999);

    vm.startPrank(vars.network.addr);
    votingPowerProvider.registerOperatorVault(operator.addr, operatorVault);
    vm.stopPrank();

    address[] memory operatorVaults = votingPowerProvider.getOperatorVaults(operator.addr);
    uint256 actualVotingPower = votingPowerProvider.getOperatorVotingPower(operator.addr,
    ↪  operatorVaults[0], "");

    console.log("30-decimal token test:");
    console.log("  Stake: 999,999 units (substantial economic value)");
    console.log("  Divisor: 1,000,000 (10^6)");
    console.log("  Actual Voting Power: %d", actualVotingPower);
    console.log("  This demonstrates COMPLETE voting power loss for legitimate stakeholders!");

    // This will fail, demonstrating the precision loss
    assertEq(actualVotingPower, 0, "Voting power is 0 due to precision loss - this is the BUG!");
}
```

Recommended Mitigation: Consider restricting tokens with > 24 decimals. Alternatively, revise the `_normalizeVaultTokenDecimals` to ensure non-zero stakes get minimum 1 voting power:

```
function _normalizeVaultTokenDecimals(address vault, uint256 votingPower) internal view virtual returns
↪  (uint256) {
    uint8 tokenDecimals = IERC20Metadata(_getCollateral(vault)).decimals();

    if (tokenDecimals > BASE_DECIMALS) {
        uint256 scaled = votingPower.scale(tokenDecimals, BASE_DECIMALS);
        return (scaled == 0 && votingPower > 0) ? 1 : scaled; //@audit give minimum voting power
    }

    return votingPower.scale(tokenDecimals, BASE_DECIMALS);
}
```

Symbiotic: Acknowledged.

Cyfrin: Acknowledged.

7.1.3 Incomplete validation in `OpNetVaultAutoDeploy` allows invalid burner configuration for registering operators with auto vault deployments

Description: `OpNetVaultAutoDeploy._validateConfig()` is missing burner address validation that could lead to deployment failures. The function fails to validate a configuration scenario where `isBurnerHook` is enabled with burner as `address(0)`.

This validation gap allows administrators to set invalid configurations that pass validation but cause all subsequent vault deployments to fail during slasher initialization. The validation logic only checks if burner hooks are enabled without slashers, but fails to validate the case where slashers are enabled with burner hooks but the burner address is zero.

```
//OpNetVaultAutoDeployLogic

 function _validateConfig(
        IOpNetVaultAutoDeploy.AutoDeployConfig memory config
    ) public view {
        if (config.collateral == address(0)) {
            revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidCollateral();
        }
        if (config.epochDuration == 0) {
```

```
            revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidEpochDuration();
        }
        uint48 slashingWindow = IVotingPowerProvider(address(this)).getSlashingWindow();
        if (config.epochDuration < slashingWindow) {
            revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidEpochDuration();
        }
        if (!config.withSlasher && slashingWindow > 0) {
            revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidWithSlasher();
        }
        if (!config.withSlasher && config.isBurnerHook) {
            revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidBurnerHook();
        } //@audit missing checks on burner address
    }
```

Here is the initialization logic in Symbiotic core's `BaseSlasher`

```
//BaseSlasher
  function _initialize(
        bytes calldata data
    ) internal override {
        (address vault_, bytes memory data_) = abi.decode(data, (address, bytes));

        if (!IRegistry(VAULT_FACTORY).isEntity(vault_)) {
            revert NotVault();
        }

        __ReentrancyGuard_init();

        vault = vault_;

        BaseParams memory baseParams = __initialize(vault_, data_);

        if (IVault(vault_).burner() == address(0) && baseParams.isBurnerHook) {
            revert NoBurner();
        } //@audit if no burner, vault deployment reverts

        isBurnerHook = baseParams.isBurnerHook;
    }
```

As a result, the following (mis)configuration is undetected while setting configuration:

```
AutoDeployConfig memory maliciousConfig = AutoDeployConfig({
    epochDuration: 1000,
    collateral: validToken,
    burner: address(0),     //  Zero address burner
    withSlasher: true,      //  Slasher enabled
    isBurnerHook: true      //  Hook enabled but no burner!
});

// @audit This passes validation incorrectly
deployer.setAutoDeployConfig(maliciousConfig);
```

Impact: All new operator registrations fail in auto-deployment mode until configuration is fixed.

Proof of Concept: Add the following tests to `OpNetVaultAutoDeploy.t.sol`

```
    function test_SetAutoDeployConfig_InvalidBurnerAddress_WithSlasherAndBurnerHook() public {
    // @audit This configuration should fail validation but currently passes
    // Config with slasher enabled, burner hook enabled, but zero address burner
    IOpNetVaultAutoDeploy.AutoDeployConfig memory maliciousConfig =
    ↪  IOpNetVaultAutoDeploy.AutoDeployConfig({
        epochDuration: slashingWindow,
```

```
        collateral: validConfig.collateral,
        burner: address(0),      // Zero address burner - THIS IS THE ISSUE
        withSlasher: true,       // Slasher enabled
        isBurnerHook: true       // Hook enabled but no valid burner!
    });

    // Currently this passes validation incorrectly - it should revert
    deployer.setAutoDeployConfig(maliciousConfig);

    // Verify the incorrect config was set
    IOpNetVaultAutoDeploy.AutoDeployConfig memory setConfig = deployer.getAutoDeployConfig();
    assertEq(setConfig.burner, address(0));
    assertTrue(setConfig.withSlasher);
    assertTrue(setConfig.isBurnerHook);
}



function test_AutoDeployFailure_WithInvalidBurnerConfig() public {
    // Set up the incorrect configuration that passes validation but causes deployment failures
    IOpNetVaultAutoDeploy.AutoDeployConfig memory maliciousConfig =
    ↪  IOpNetVaultAutoDeploy.AutoDeployConfig({
        epochDuration: slashingWindow,
        collateral: validConfig.collateral,
        burner: address(0),      // Zero address burner
        withSlasher: true,       // Slasher enabled
        isBurnerHook: true       // Hook enabled
    });

    // This should pass validation
    deployer.setAutoDeployConfig(maliciousConfig);

    // Enable auto deployment
    deployer.setAutoDeployStatus(true);

    // Now try to register an operator - this should fail during vault deployment
    // because BaseSlasher._initialize() will revert with NoBurner() when it finds
    // that burner is address(0) but isBurnerHook is true
    vm.startPrank(operator1);

    // @audit This registration should fail during auto-deployment due to slasher initialization
    vm.expectRevert();
    deployer.registerOperator();

    vm.stopPrank();

    // Verify no vault was deployed
    address deployedVault = deployer.getAutoDeployedVault(operator1);
    assertEq(deployedVault, address(0));

    // Verify operator has no vaults
    address[] memory vaults = deployer.getOperatorVaults(operator1);
    assertEq(vaults.length, 0);
}
```

Recommended Mitigation: Consider adding following check to _validateConfig:

```
function _validateConfig(
    IOpNetVaultAutoDeploy.AutoDeployConfig memory config
) public view {
    // ... existing validations ...

    // @audit validate burner address
```

```
    if (config.withSlasher && config.isBurnerHook && config.burner == address(0)) {
        revert IOpNetVaultAutoDeploy.OpNetVaultAutoDeploy_InvalidBurnerAddress();
    }
}
```

Symbiotic: Fixed in 625ea22.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Missing bounds check on weight values in `WeightedTokensVPCalc` and `WeightedVaultsVPCalc`

Description: The `WeightedTokensVPCalc` and `WeightedVaultsVPCalc` contracts lack proper bounds checking when setting weight values, which could lead to integer overflow during voting power calculations or complete elimination of voting power through zero weights.

The weight-setting functions in both contracts accept any `uint208` value without validation:

```
// WeightedTokensVPCalc.sol
function setTokenWeight(address token, uint208 weight) public virtual checkPermission {
    _setTokenWeight(token, weight);  // No bounds checking
}

// WeightedVaultsVPCalc.sol
function setVaultWeight(address vault, uint208 weight) public virtual checkPermission {
    _setVaultWeight(vault, weight);  // No bounds checking
}
```

The voting power calculation multiplies stake amounts by these weights without overflow protection:

```
// WeightedTokensVPCalc.sol
function stakeToVotingPower(address vault, uint256 stake, bytes memory extraData)
    public view virtual override returns (uint256) {
    return super.stakeToVotingPower(vault, stake, extraData) * getTokenWeight(_getCollateral(vault));
    ↪  //@audit could go to 0 or overflow based on weight set
 }
```

Impact: Cause an integer overflow (extremely large weight) or total domination of one token over the rest or complete voting power elimination (0 weight).

While the weight-setting functions are protected by the `checkPermission` modifier and controlled by network governance in production deployments, technical safeguards remain important.

Recommended Mitigation: Consider implementing a min and max weight that are either constants or immutable.

```
contract WeightedTokensVPCalc is NormalizedTokenDecimalsVPCalc, PermissionManager {
    uint208 public constant MIN_WEIGHT = 1e6;      // Minimum non-zero weight
    uint208 public constant MAX_WEIGHT = 1e18;     // Maximum reasonable weight

    function setTokenWeight(address token, uint208 weight) public virtual checkPermission {
        require(weight <= MAX_WEIGHT, "Weight exceeds maximum");
        require(weight <= MAX_SAFE_WEIGHT, "Weight risks overflow");

        _setTokenWeight(token, weight);
    }
}
```

Symbiotic: Fixed in 2a8b18d.

Cyfrin: Verified.

7.2.2 `unwhitelistOperator` allows state changes when whitelist Is disabled, causing inconsistent operator state

Description: The `unwhitelistOperator` function performs state changes (removing an operator from the whitelist and potentially unregistering them) even when the whitelist feature is **disabled**. This violates the expected behavior that **whitelist enforcement should only be active when explicitly enabled**.

Unwhitelisting an operator while the whitelist is disabled silently alters the contract's state. Later, when the whitelist is re-enabled, the operator is unexpectedly no longer whitelisted — even though no whitelist-related logic was supposed to be active when they were removed.

Impact: When the whitelist feature is disabled, `unwhitelistOperator` silently processes changes, leading to an inconsistent state. If the whitelist is re-enabled, an operator who was unwhitelisted, remains registered, leading to inconsistent state.

Recommended Mitigation: Consider preventing `unwhitelistOperator` from executing when whitelist is disabled. Either revert explicitly or skip execution:

```
function unwhitelistOperator(address operator) public virtual checkPermission {
    if (!isWhitelistEnabled()) {
        revert OperatorsWhitelist_WhitelistDisabled( );
    }

    _getOperatorsWhitelistStorage()._whitelisted[operator] = false;

    if (isOperatorRegistered(operator)) {
        _unregisterOperator(operator);
    }

    emit UnwhitelistOperator(operator);
}
```

Symbiotic: Fixed in 32bea5e.

Cyfrin: Verified.

7.3 Informational

7.3.1 `OpNetVaultAutoDeployLogic::getVetoSlasherParams` is never used

Description: `OpNetVaultAutoDeployLogic::getVetoSlasherParams` is dead code that is never used. It is also worth noting that `OpNetVaultAutoDeploy` is designed only for `InstantSlasher` and has no mechanism to utilize `VetoSlasher` functionality.

Recommended Mitigation: Consider removing the function `getVetoSlasherParams` from the `OpNetVaultAutoDeployLogic` library.

Symbiotic: Acknowledged. Unchanged to simplify external customizations

Cyfrin: Acknowledged.

7.3.2 `__NoncesUpgradeable_init` is not invoked

Description: The `__NoncesUpgradeable_init` function from the `NoncesUpgradeable` contract is not being called during the initialization of the `VotingPowerProvider` abstract contract. Since `VotingPowerProvider` inherits from `NoncesUpgradeable`, failing to initialize the nonce-related state can lead to incorrect nonce management, potentially causing issues with replay protection in signature verification or other related functionalities.

Recommended Mitigation: Update the `__VotingPowerProvider_init` function to include a call to `__NoncesUpgradeable_init()`

```
function __VotingPowerProvider_init(
VotingPowerProviderInitParams memory votingPowerProviderInitParams
) internal virtual onlyInitializing {
__NetworkManager_init(votingPowerProviderInitParams.networkManagerInitParams);
VotingPowerProviderLogic.initialize(votingPowerProviderInitParams);
__OzEIP712_init(votingPowerProviderInitParams.ozEip712InitParams);
__NoncesUpgradeable_init(); // @audit Add this line to properly initialize nonce state
}
```

Symbiotic: Acknowledged. Not invoked to optimize bytecode size. Should be safe since it's not initializing any state inside

Cyfrin: Acknowledged.

7.3.3 Redundant manual access control checks

Description: In both `BaseRewards` and `BaseSlashing` contracts, functions perform manual access control checks (`_checkRewarder` and `_checkSlasher`) instead of using the already-defined `onlyRewarder` and `onlySlasher` modifiers. Specifically:

- `distributeStakerRewards` and `distributeOperatorRewards` in `BaseRewards`
- `slashVault` and `executeSlashVault` in `BaseSlashing`

This inconsistency reduces code readability and increases the risk of missing or duplicating access control logic in the future.

Recommended Mitigation: Replace manual `_checkRewarder` and `_checkSlasher` calls with their corresponding modifiers for cleaner and more consistent access control enforcement:

```
function distributeStakerRewards(...) public virtual onlyRewarder { ... }

function distributeOperatorRewards(...) public virtual onlyRewarder { ... }

function slashVault(...) public virtual onlySlasher returns (...) { ... }

function executeSlashVault(...) public virtual onlySlasher returns (...) { ... }
```

This ensures access checks are declarative, standardized, and easier to maintain.

Symbiotic: Acknowledged. Intended to decrease bytecode size.

Cyfrin: Acknowledged.

7.3.4 Unused functions in `KeyRegistry`

Description: In the `KeyRegistry` contract, there are three internal methods designed to handle 64-byte key operations:

- `_setKey64(address operator, uint8 tag, bytes memory key)`
- `_getKey64At(address operator, uint8 tag, uint48 timestamp)`
- `_getKey64(address operator, uint8 tag)`

However, none of these functions are ever invoked in the current contract implementation.

Recommended Mitigation: Remove the unused methods (`_setKey64`, `_getKey64`, `_getKey64At`) to improve code clarity, reduce audit surface area, and eliminate potential dead code.

Symbiotic: Acknowledged. Intended for future customizations.

Cyfrin: Acknowledged.

7.3.5 Inconsistent error handling for empty data

Description: In `_getSelector()`, when data is empty, it returns `0xEEEEEEEE` which is not a standard practice.

```
function _getSelector(bytes memory data) internal pure returns (bytes4 selector) {
    if (data.length == 0) {
        return 0xEEEEEEEE;
    }
    // ...
}
```

Impact: Potentially unexpected return values confusing functions interacting with the `_getSelector` function.

Recommended Mitigation: Consider documenting the chosen behaviour so developers and future auditors know whether it is expected behaviour.

Symbiotic: Fixed in 8667780.

Cyfrin: Verified.

7.4 Gas Optimization

7.4.1 `PersistentSet` library storage access

Description: Multiple expensive storage reads (SLOAD ~2100 gas each) for the same `set._statuses[value]` occur in several functions: `_add`, `_remove`, `_containsAt`, and `_contains`. Each repeated access to the same storage slot wastes gas.

Recommended Mitigation: Cache storage pointers to avoid repeated SLOADs in all affected functions:

```
function _add(Set storage set, uint48 key, bytes32 value) private returns (bool) {
    unchecked {
        Status storage status = set._statuses[value]; // Cache once
        if (status.isAdded) {
            if (status.isRemoved.latest() == 0) {
                return false;
            }
            status.isRemoved.push(key, 0);
        } else {
            set._elements.push(value);
            status.isAdded = true;
            status.addedAt = key;
        }
        set._length += 1;
        return true;
    }
}

function _containsAt(Set storage set, uint48 key, bytes32 value, bytes memory hint) private view
↪   returns (bool) {
    Status storage status = set._statuses[value]; // Cache once
    return status.isAdded && key >= status.addedAt
        && status.isRemoved.upperLookupRecent(key, hint) == 0;
}

function _contains(Set storage set, bytes32 value) private view returns (bool) {
    Status storage status = set._statuses[value]; // Cache once
    return status.isAdded && status.isRemoved.latest() == 0;
}
```

This saves ~2100-4200 gas per function call by eliminating redundant storage reads.

Symbiotic: Fixed in dcb1d5a.

Cyfrin: Verified.

7.4.2 Cache storage reads in `upperLookupRecentCheckpoint`

Description: Functions like `upperLookupRecentCheckpoint` in `Checkpoints` perform multiple expensive storage reads for `self._trace._checkpoints.length` and repeated `_unsafeAccess` calls.

Recommended Mitigation: Cache the checkpoints array reference and length:

```
function upperLookupRecentCheckpoint(
    Trace208 storage self,
    uint48 key
) internal view returns (bool, uint48, uint208, uint32) {
    OZCheckpoints.Checkpoint208[] storage checkpoints = self._trace._checkpoints; // Cache
    uint256 len = checkpoints.length; // Cache length

    uint256 low = 0;
    uint256 high = len;

    if (len > 5) {
```

```
        uint256 mid = len - Math.sqrt(len);
        if (key < _unsafeAccess(checkpoints, mid)._key) { // Use cached reference
            high = mid;
        } else {
            low = mid + 1;
        }
    }

    uint256 pos = _upperBinaryLookup(checkpoints, key, low, high);
    // ... rest of function
}
```

Symbiotic: Acknowledged.

Cyfrin: Acknowledged.

7.4.3 Loop can use unchecked arithmetic and cache array length

Description: The `Network::scheduleBatch()` function has multiple loops that can be optimized with unchecked arithmetic and cached array lengths.

Recommended Mitigation:

```
function scheduleBatch(/*...*/) public virtual override onlyRole(PROPOSER_ROLE) {
    uint256 targetsLength = targets.length;
    if (targetsLength != values.length || targetsLength != payloads.length) {
        revert TimelockInvalidOperationLength(targetsLength, payloads.length, values.length);
    }

    unchecked {
        for (uint256 i; i < targetsLength; ++i) {
            uint256 minDelay = getMinDelay(targets[i], payloads[i]);
            if (delay < minDelay) {
                revert TimelockInsufficientDelay(delay, minDelay);
            }
        }
    }

    bytes32 id = hashOperationBatch(targets, values, payloads, predecessor, salt);
    _scheduleOverriden(id, delay);

    unchecked {
        for (uint256 i; i < targetsLength; ++i) {
            emit CallScheduled(id, i, targets[i], values[i], payloads[i], predecessor, delay);
        }
    }
    // ...
}
```

Symbiotic: Acknowledged.

Cyfrin: Acknowledged.