Wonderland - Optimism Fee Splitter

Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher
Haxatron, Lead Security Researcher

November 4, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

From Oct 15th to Oct 19th the Cantina team conducted a review of optimism on commit hash 989afa81. The review focused on the following scope:

- All elements highlighted in the following notion page: `https://oplabs.notion.site/FeeSplitter-262f153ee1628034942fe1c10ed51fb0`.

The team identified a total of **42** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	10	7	3
Gas Optimizations	3	2	1
Informational	29	22	7
Total	**42**	**31**	**11**

The Cantina Managed team conducted a holistic review of Optimism's `optimism` within the scope described on their notion page, at commit hash f1fcd964, and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Vault minimum withdrawal amount is set nonzero for newly deployed chains starting from genesis

Severity: Low Risk

Context: L2Genesis.s.sol#L311-L333, L2Genesis.s.sol#L401-L423, L2Genesis.s.sol#L425-L447, L2Genesis.s.sol#L449-L471

Description: The new `FeeSplitter` setup requires all configured `FeeVaults` to have their `minWithdrawalAmount` set to 0. This is to prevent one fee vault from blocking the entire fee collection in case it doesn't meet the threshold, as explained by the following inline comment.

FeeVault.sol#L103-L104

```
/// @dev If integrating the FeeSplitter contract, the minimum withdrawal amount must be set to 0 to
/// avoid blocking withdrawals and disbursements for all vaults if one vault doesn't reach the threshold.
```

However, this is not the case for newly deployed chains starting from genesis using the default configuration (with no non-default overrides specified). When deploying with the `op-deployer`, the `*FeeVaultMinimumWithdrawalAmount` passed to the `L2Genesis.s.sol` deployment script uses `standard.VaultMinWithdrawalAmount` as the default value and `UseRevenueShare` to true (meaning that `FeeSplitter` configuration will be set up on default).

l2genesis.go#L166-L181

```
func defaultOverrides() l2GenesisOverrides {
    return l2GenesisOverrides{
    FundDevAccounts:                          false,
    BaseFeeVaultMinimumWithdrawalAmount:      standard.VaultMinWithdrawalAmount,
    L1FeeVaultMinimumWithdrawalAmount:        standard.VaultMinWithdrawalAmount,
    SequencerFeeVaultMinimumWithdrawalAmount: standard.VaultMinWithdrawalAmount,
    OperatorFeeVaultMinimumWithdrawalAmount:  standard.VaultMinWithdrawalAmount,
    BaseFeeVaultWithdrawalNetwork:            "local",
    L1FeeVaultWithdrawalNetwork:              "local",
    SequencerFeeVaultWithdrawalNetwork:       "local",
    OperatorFeeVaultWithdrawalNetwork:        "local",
    EnableGovernance:                         false,
    GovernanceTokenOwner:                     standard.GovernanceTokenOwner,
    UseRevenueShare:                          true,
    }
}
```

standard.go#L54

```
var VaultMinWithdrawalAmount = mustHexBigFromHex("0x8ac7230489e80000")
```

However `standard.VaultMinWithdrawalAmount` is set to non-zero, 10 ether to be exact. Then during deployment via `L2Genesis.s.sol`, for all 4 vaults, when `useRevenueShare` is set to true, the deploy script uses `_input.*FeeVaultMinimumWithdrawalAmount` which is the value passed by `op-deployer` instead of using 0 as the value. Using the `SequencerFeeVault` deployment as an example:

L2Genesis.s.sol#L311-L333

```
/// @notice This predeploy is following the safety invariant #2,
function setSequencerFeeVault(Input memory _input) internal {
    address recipient;
    Types.WithdrawalNetwork network;
    if (_input.useRevenueShare) {
        recipient = Predeploys.FEE_SPLITTER;
        network = Types.WithdrawalNetwork.L2;
    } else {
        recipient = _input.sequencerFeeVaultRecipient;
        network = Types.WithdrawalNetwork(_input.sequencerFeeVaultWithdrawalNetwork);
    }

    address impl = _setImplementationCode(Predeploys.SEQUENCER_FEE_WALLET);

    /// Initialize the implemenation using max value for min withdrawal amount to make it unusable
```

```
    ISequencerFeeVault(payable(impl)).initialize(address(0), type(uint256).max, Types.WithdrawalNetwork.L1);
    // Initialize the predeploy
    ISequencerFeeVault(payable(Predeploys.SEQUENCER_FEE_WALLET)).initialize({
        _recipient: recipient,
        _minWithdrawalAmount: _input.sequencerFeeVaultMinimumWithdrawalAmount,
        _withdrawalNetwork: network
    });
}
```

This was confirmed and verified on our own local chain where we observed that the configured `minWithdrawalAmount` was incorrectly set to 10 ether.

As such for newly deployed chains, they might run into an issue where one fee vault can block the entire fee collection in case it doesn't meet the 10 ether threshold. However, this can be easily resolved by the proxy admin using `setMinWithdrawalAmount` to resolve the issue by setting all the `minWithdrawalAmount` to 0, so the result is a temporary DOS of fee collection.

Recommendation: In `L2Genesis.s.sol`, if `_input.useRevenueShare` is true set the `minWithdrawalAmount` to 0 for all 4 vaults. Alternatively in `FeeSplitter`, `disburseFees()` / `_feeVaultWithdrawal()` use a try/catch around the call to `withdraw()` and ignore reverts.

Optimism: Fixed in PR 646.

Cantina Managed: Fix verified.

3.1.2 Different and old OpenZeppelin libraries

Severity: Low Risk

Context: FeesDepositor.sol#L8, FeesDepositor.sol#L8-L10, FeeSplitter.sol#L16, FeeSplitter.sol#L16-L18, FeeVault.sol#L15

Description: The contracts `FeeSplitter` and `FeesDepositor` use old libraries: OZ v4.7.1 and OZ v4.7.1 upgradable

Note: both could use use the same type of library.

The `FeeVault` uses a newer version OZ v5.0.2, although that version is already more than 1.5 years old. The main advantage of the `OZ V5` library is that the storage variables use the eip-7201 namespaced storage layout, which makes upgrading easier. Older libraries could have bugs that are fixed in newer versions.

Recommendation: Consider using the same `OZ V5` library for all three contracts, preferably a more recent version.

Optimism: Acknowledged. Created issue oz-dependencies-consistency for tracking.

Cantina Managed: Acknowledged.

3.1.3 No emit in `FeeSplitter::initialize()`

Severity: Low Risk

Context: IFeesDepositor.sol#L10, IBaseFeeVault.sol#L12, IFeeSplitter.sol#L8, IFeeVault.sol#L11, IL1FeeVault.sol#L12, IOperatorFeeVault.sol#L12, ISequencerFeeVault.sol#L12, FeeSplitter.sol#L107-L111

Description: According to the FeeSplitter specification, the function `initialize()` must emit an `Initialized` event with the provided parameters. However this isn't present. The `Initialized` events in the following interface files are not used:

- `IFeesDepositor`.
- `IFeeSplitter`.
- `IFeeVault`.
- `IBaseFeeVault`.
- `IL1FeeVault`.
- `IOperatorFeeVault`.
- `ISequencerFeeVault`.

Most `Initialized` events have an `uint64` parameters. However the events of `IFeeSplitter` and `IFeesDepositor` use an `uint8`.

Recommendation: Consider adding the emit in `FeeSplitter::initialize()` and also doublecheck all the other`Initialized` events.

Optimism: Fixed in commit b81d1cf4.

Cantina Managed: Fix verified.

3.1.4 More strict checks in `receive()`

Severity: Low Risk

Context: FeeSplitter.sol#L114-L123, FeeSplitter.sol#L126-L140, FeeSplitter.sol#L211-L219, FeeSplitter.sol#L223-L235

Description: The `FeeSplitter design document` has the following line for invariant 5:

> When not on a disbursement context, `receive` MUST revert (`FeeVault.withdraw()` MUST revert if not call from the `FeeSplitter` when configured as its recipient).

This is implemented in `FeeSplitter::receive()`. However there is an edge case: if one of the vaults is changed and is able to (indirectly) call a `withdraw()` on a different vault, it will still work because `_isTransientDisbursing() == true`.

Recommendation: If you want to make the check more strict consider changing the code to something like this:

```
receive() external payable virtual {
    if (msg.sender != _getTransientDisbursingAddress() )
        revert FeeSplitter_SenderNotApprovedVault();
    emit FeesReceived({ sender: msg.sender, amount: msg.value });
}
function _feeVaultWithdrawal(address payable _feeVault) internal returns (uint256 value_) {
    // ...
    _setTransientDisbursingAddress( address(_feeVault) );
    value_ = IFeeVault(_feeVault).withdraw();
}
function _setTransientDisbursingAddress(address allowedCaller_) internal {
    assembly {
        tstore(_FEE_SPLITTER_DISBURSING_ADDRESS_SLOT, allowedCaller_)
    }
}
function _getTransientDisbursingAddress() internal view returns (address allowedCaller_) {
    assembly {
        allowedCaller_ := tload(_FEE_SPLITTER_DISBURSING_ADDRESS_SLOT)
    }
}
```

Optimism: Fixed in PR 647.

Cantina Managed: Fix verified.

3.1.5 Function `disburseFees()` doesn't check the fees have been received

Severity: Low Risk

Context: FeeSplitter.sol#L126-L142

Description: Function `disburseFees()` doesn't explicitly check the fees have actually been received. If a vault incorrectly reports that amount, that amount is still used.

Note: the code will revert lateron if insufficient native token is present to be distributed so the risk is minimal, however that might be more difficult to troubleshoot.

Recommendation: Consider explicitly checking the fees have been received.

3.1.6 Function `disburseFees()` doesn't use any native token that is already present in the contract

Severity: Low Risk

Context: FeeSplitter.sol#L126-L152, FeeSplitter.sol#L173-L174

Description: Function `disburseFees()` doesn't use any native token that is already present in the contract. These tokens could end up there in the following ways:

- Via selfdestruct / Solady SafeTransferLib::forceSafeTransferETH().
- Potentially in the future via the new opcode GAS2ETH.

The native tokens will always stay in the contract, although they can be rescued via upgrading the `FeeSplitter`.

Note: precursors of this contract: base and unichain do use the balance.

Recommendation: Consider also using the present native tokens.

Optimism: Acknowledged. This is a design choice. We want to only use the fees pulled from the vaults for the revenue calculation to avoid an edge case where someone sends balance to game the math.

Cantina Managed: Acknowledged.

3.1.7 Withdrawal transactions can fail (but can be replayed later) due to resource metering on deposit transactions

Severity: Low Risk

Context: FeesDepositor.sol#L84-L94

Description: During the L2 -> L1 flow when the fees are sent to the L1, `FeeDepositor.receive` is called within `OptimismPortal.finalizeWithdrawalTransaction -> CrossDomainMessenger.relayMessage`. This `receive` initiates another `CrossDomainMessenger.sendMessage -> OptimismPortal.depositTransaction` to send the received fees to OP mainnet chain, when the `minDepositAmount` is reached.

FeesDepositor.sol#L84-L94

```
/// @notice Receives ETH and sends it to the L2 recipient via CrossDomainMessenger when the threshold is reached.
receive() external payable {
    uint256 balance = address(this).balance;
    emit FundsReceived(msg.sender, msg.value, balance);

    if (balance >= minDepositAmount) {
        address recipient = l2Recipient;
        messenger.sendMessage{ value: balance }(recipient, hex"", gasLimit);
        emit FeesDeposited(recipient, balance);
    }
}
```

In this flow, the `CrossDomainMessenger.sendMessage -> OptimismPortal.depositTransaction` can fail due to resource metering. Resource metering is where the Optimism Portal applies EIP-1559 style gas pricing to burn L1 gas in exchange for L2 gas when deposit transactions are executed.

ResourceMetering.sol#L77-L151

```
/// @notice An internal function that holds all of the logic for metering a resource.
/// @param _amount     Amount of the resource requested.
/// @param _initialGas The amount of gas before any modifier execution.
function _metered(uint64 _amount, uint256 _initialGas) internal {
    // Update block number and base fee if necessary.
    uint256 blockDiff = block.number - params.prevBlockNum;

    ResourceConfig memory config = _resourceConfig();
    int256 targetResourceLimit =
        int256(uint256(config.maxResourceLimit)) / int256(uint256(config.elasticityMultiplier));

    if (blockDiff > 0) {
        // Handle updating EIP-1559 style gas parameters. We use EIP-1559 to restrict the rate
        // at which deposits can be created and therefore limit the potential for deposits to
        // spam the L2 system. Fee scheme is very similar to EIP-1559 with minor changes.
        int256 gasUsedDelta = int256(uint256(params.prevBoughtGas)) - targetResourceLimit;
        int256 baseFeeDelta = (int256(uint256(params.prevBaseFee)) * gasUsedDelta)
```

```
            / (targetResourceLimit * int256(uint256(config.baseFeeMaxChangeDenominator)));

        // Update base fee by adding the base fee delta and clamp the resulting value between
        // min and max.
        int256 newBaseFee = Arithmetic.clamp({
            _value: int256(uint256(params.prevBaseFee)) + baseFeeDelta,
            _min: int256(uint256(config.minimumBaseFee)),
            _max: int256(uint256(config.maximumBaseFee))
        });

        // If we skipped more than one block, we also need to account for every empty block.
        // Empty block means there was no demand for deposits in that block, so we should
        // reflect this lack of demand in the fee.
        if (blockDiff > 1) {
            // ...
        }

        // Update new base fee, reset bought gas, and update block number.
        params.prevBaseFee = uint128(uint256(newBaseFee));
        params.prevBoughtGas = 0;
        params.prevBlockNum = uint64(block.number);
    }

    // Make sure we can actually buy the resource amount requested by the user.
    params.prevBoughtGas += _amount;
    if (int256(uint256(params.prevBoughtGas)) > int256(uint256(config.maxResourceLimit))) {
        revert OutOfGas();
    }

    // Determine the amount of ETH to be paid.
    uint256 resourceCost = uint256(_amount) * uint256(params.prevBaseFee);

    // We currently charge for this ETH amount as an L1 gas burn, so we convert the ETH amount
    // into gas by dividing by the L1 base fee. We assume a minimum base fee of 1 gwei to avoid
    // division by zero for L1s that don't support 1559 or to avoid excessive gas burns during
    // periods of extremely low L1 demand. One-day average gas fee hasn't dipped below 1 gwei
    // during any 1 day period in the last 5 years, so should be fine.
    uint256 gasCost = resourceCost / Math.max(block.basefee, 1 gwei);

    // Give the user a refund based on the amount of gas they used to do all of the work up to
    // this point. Since we're at the end of the modifier, this should be pretty accurate. Acts
    // effectively like a dynamic stipend (with a minimum value).
    uint256 usedGas = _initialGas - gasleft();
    if (gasCost > usedGas) {
        Burn.gas(gasCost - usedGas);
    }
}
```

To simplify the above, deposit transactions from L1 -> L2 are metered much like regular transactions to prevent users from spamming the L2 block space. EIP-1559 style pricing is applied to deposit transactions where the gas price of a deposit transaction is determined and adjusted based on the current gas usage of deposit transactions from previous blocks. L1 gas is then burnt via `Burn.gas` to pay for the L2 gas.

As such there are two scenarios where the `FeeDepositor.receive` can fail:

1. When not enough gas is supplied to meet the dynamic gas required to be burnt to pay for the deposit transaction or...
2. When adding the deposit transaction requests more gas then can be fit into the per-block maximum resource limit in `if (int256(uint256(params.prevBoughtGas)) > int256(uint256(config.maxResourceLimit))) {`.

This would be a High severity issue due to the potential for locked funds when the withdrawal transaction fails. But fortunately, the L2 $\rightarrow$ L1 flow makes use of `CrossDomainMessenger`s, so in case the withdrawal transaction fails, the `relayMessage` function can be called again with higher gas limits to replay the withdrawal message. Hence, in the worst-case scenario, this issue would only lead to a waste of gas to replay the message from the `CrossDomainMessenger`.

Recommendation: At the minimum, acknowledge and possibly document that the use of `CrossDomainMessenger` is important in mitigating the above scenario and should not be removed from the above flow. If possible, it would be good to separate the L2 $\rightarrow$ L1 flow to a two-step process where `receive` only handles ETH, and another function that anyone can call and can be used to initiate the L1 $\rightarrow$ L2 flow to send the fees from the `FeesDepositor` to the OP Mainnet recipient, however this would come at a tradeoff for

convenience.

Optimism: Fixed in commit b81d1cf4.

Cantina Managed: Fix verified.

3.1.8 `_WITHDRAWAL_MIN_GAS` of `FeeVault` is not configurable

Severity: Low Risk

Context: FeeVault.sol#L20-L25, L1Withdrawer.sol#L106-L116

Description: `_WITHDRAWAL_MIN_GAS` of `FeeVault` is not configurable, however the similar value in `L1Withdrawer` is configurable via `setWithdrawalGasLimit`. In a comment `L1Withdrawer` indicated a higher value is required.

```
/// @dev If target on L1 is `FeesDepositor`, the gas limit should be above 800k gas.
```

Recommendation: Consider making the `_WITHDRAWAL_MIN_GAS` of `FeeVault` configurable.

Optimism: Acknowledged. After discussing it internally, we think moving forward with this change is not the way to go. Firstly, this is not part of the scope of the revenue sharing project, this is previous logic from the FeeVault that we don't need to change. Also, the FeesDepositor is not intended to be used to withdraw the whole balance from the vault, but only the Optimism corresponding share of the fees. This path is outside of the system and we shouldn't touch it, but probably it is there to withdraw the whole balance to an EOA due to the small min gas limit constant.

Cantina Managed: Acknowledged.

3.1.9 Not all predeploys are present in `getAddress()`

Severity: Low Risk

Context: Artifacts.s.sol#L67-L71, Artifacts.s.sol#L91-L92, Artifacts.s.sol#L115-L120

Description: Function `getAddress()` allow to query predeploys by name. However predeploy `FeeSplitter` / `FEE_SPLITTER` is missing.

Recommendation: Consider adding the following to `getAddress()`:

```
else if (digest == keccak256(bytes("FeeSplitter"))) {
    return payable(Predeploys.FEE_SPLITTER);
} ...
```

Optimism: Fixed in PR 646.

Cantina Managed: Fix verified.

3.1.10 Reentrancy possible in `disburseFees()`

Severity: Low Risk

Context: FeeSplitter.sol#L126, FeeSplitter.sol#L126-L132, FeeSplitter.sol#L166-L169

Description: A reentrancy is possible in `disburseFees()`, because it does external calls via `SafeCall.send()`. A reentrant call to `disburseFees()` could be executed if `feeDisbursementInterval == 0` and a small amount of native token is send to one of the vaults. The risk of this of seems limited, although the order of the emits could be confusing.

The FeeSplitter specification mentions a `nonReentrant` pattern, although it isn't explicitly specified for `disburseFees()`.

Recommendation: Consider enforcing `feeDisbursementInterval > 0` and/or consider adding a `nonReentrant` modifier.

Optimism: Fixed in PR 648.

Cantina Managed: Fix verified.

3.2 Gas Optimization

3.2.1 For loop can be optimized

Severity: Gas Optimization

Context: FeeSplitter.sol#L126, FeeSplitter.sol#L159

Description: The for loop in `disburseFees()` can be optimized by caching the array length.

Recommendation: Consider caching the array length in the for loop.

Optimism: Fixed in PR 649.

Cantina Managed: Fix verified.

3.2.2 Emits for setters can be optimized

Severity: Gas Optimization

Context: FeeVault.sol#L118-L127

Description: All the set functions emit an event. These functions can be optimized to save some gas.

Recommendation: Consider changing the set functions in the following way:

```
 function setXyz(address _newXyz) external {
      // ...
+     emit XyzUpdated(xyz, _newXyz); // use the old value before its updated
-     address oldXyz = xyz;
      xyz = _newXyz;
-     emit XyzUpdated(oldXyz, _newXyz);

 }
```

Optimism: Acknowledged. Leaning towards leaving as is to emit the events at the end and keep it consistent.

Cantina Managed: Acknowledged.

3.2.3 Function `withdraw()` can be optimized

Severity: Gas Optimization

Context: FeeVault.sol#L146-L168

Description: The value of storage variable `recipient` is used multiple times in `withdraw()`. Some gas could be saved by caching this.

Recommendation: Consider caching the value of `recipient`.

Optimism: Fixed in PR 649.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Namespaced storage layout could be used

Severity: Informational

Context: FeesDepositor.sol#L15-L24, FeeSplitter.sol#L71-L78, FeeVault.sol#L28-L40, FeeVault.sol#L39-L40

Description: The proxied contract storage variables that could be rearranged with upgrades. To reduce the impact, `_gap` is added in `FeeVault`. Recent contract frequently use the eip-7201 namespaced storage layout pattern which makes upgrading easier.

Recommendation: Consider using the eip-7201 namespaced storage layout pattern.

Note: `_gap` can then be removed.

Note: make sure to use the corresponding OZ libraries, see finding "Different and old OpenZeppelin libraries"

Optimism: Acknowledged. Created issue oz-dependencies-consistency for tracking.

Cantina Managed: Acknowledged.

3.3.2 Comment at `FeeSplitter` isn't accurate

Severity: Informational

Context: FeeSplitter.sol#L20-L22

Description: The comment at `FeeSplitter` isn't accurate because `FeeSplitter` is rather flexible. It is the `SharesCalculator` that determines the destinations.

Recommendation: Consider making the comment more generic.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.3 `lastDisbursementTime` not initialized

Severity: Informational

Context: FeeSplitter.sol#L107-L111, FeeSplitter.sol#L126-L132

Description: Function `disburseFees()` uses `lastDisbursementTime`, however this is not initialized and thus has an initial value of 0. This makes it more difficult to check the calculation in `disburseFees()` is always correct.

Recommendation: Consider setting `lastDisbursementTime = block.timestamp` in `initialize()`.

Optimism: Fixed in PR 645.

Cantina Managed: Fix verified.

3.3.4 Incorrect specification of `FeeSplitter`

Severity: Informational

Context: FeeSplitter.sol#L107-L111

Description: The FeeSplitter specification contains the following:

> MUST set `feeDisbursementInterval` to `_feeDisbursementInterval`.

However `_feeDisbursementInterval` is not defined and is a leftover from a previous version.

Recommendation: Consider changing the specification to:

```
- MUST set feeDisbursementInterval to _feeDisbursementInterval.
+ MUST set feeDisbursementInterval to 1 day.
```

Optimism: Fixed in commit b81d1cf4.

Cantina Managed: Fix verified.

3.3.5 `receive()` and `virtual`

Severity: Informational

Context: FeesDepositor.sol#L85, FeeSplitter.sol#L114, FeeVault.sol#L98, L1Withdrawer.sol#L69

Description: `FeeSplitter::receive()` is the only instance that also has `virtual`. Other functions in `FeeSplitter` don't have the `virtual` keyword.

Recommendation: Doublecheck the usefulness of `virtual` and consider removing it.

Optimism: Fixed in PR 641.

Cantina Managed: Fix verified.

3.3.6 Some predeploys are missing when inspecting semvers from `op-deployer`

Severity: Informational

Context: semvers.go#L140-L158

Description: Some predeploys are missing, when inspecting contract `semvers` from `op-deployer`, via the `inspect l2-semvers` command.

`op-deployer inspect l2-semvers --workdir .deployer 901`:

```
{
  "L2ToL1MessagePasser": "1.1.2",
  "DeployerWhitelist": "1.1.2",
  "WETH": "1.1.1",
  "L2CrossDomainMessenger": "2.2.0",
  "L2StandardBridge": "1.13.0",
  "SequencerFeeVault": "1.6.0",
  "OptimismMintableERC20Factory": "1.10.2",
  "L1BlockNumber": "1.1.2",
  "GasPriceOracle": "1.4.0",
  "L1Block": "1.6.1",
  "LegacyMessagePasser": "1.1.2",
  "L2ERC721Bridge": "1.10.0",
  "OptimismMintableERC721Factory": "1.4.2",
  "BaseFeeVault": "1.6.0",
  "L1FeeVault": "1.6.0",
  "SchemaRegistry": "1.3.1-beta.2",
  "EAS": "1.4.1-beta.3",
  "CrossL2Inbox": "",
  "L2toL2CrossDomainMessenger": "",
  "SuperchainETHBridge": "",
  "ETHLiquidity": "",
  "SuperchainTokenBridge": "",
  "OptimismMintableERC20": "1.4.1",
  "OptimismMintableERC721": "1.3.2"
}
```

More specifically, the `OperatorFeeVault` and `FeeSplitter` predeploys are missing.

Recommendation: Configure the `OperatorFeeVault` and `FeeSplitter` predeploys in `op-deployer/pkg/deployer/inspect/semvers.go`.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.7 Emit at `receive()` not the same everywhere

Severity: Informational

Context: FeesDepositor.sol#L87, FeeSplitter.sol#L122, L1Withdrawer.sol#L71

Description: The `receive()` function in `FeesDepositor` and `L1Withdrawer` also include the `balance`, however the `FeeSplitter` version doesn't do that. This is inconsistent.

Recommendation: Consider also emitting the `balance` in `FeeSplitter::receive()`.

Optimism: Fixed in PR 642.

Cantina Managed: Fix verified.

3.3.8 `disburseFees()` could check for minimum amount

Severity: Informational

Context: FeeSplitter.sol#L126, FeeSplitter.sol#L144-L147

Description: `disburseFees()` continues executing if a trivial amount of fees have been collected. In that situation the gas costs might be higher than the amount distributed.

Note: normally the `feeDisbursementInterval` also prevents small amounts being send.

Recommendation: Consider requiring a minimum amount.

Optimism: Acknowledged, but won't introduce any change here to avoid setting an arbitrary value that could be off.

Cantina Managed: Acknowledged.

3.3.9 Transient keyword

Severity: Informational

Context: FeeSplitter.sol#L221-L235

Description: The functions `_setTransientDisbursing()` and `_isTransientDisbursing()` use assemly to access transient storage. In solidity version 0.8.28 the solidity `transient` keyword is fully supported, so that can also be used. There is only one instance of transient storage in the contract, so there is no problem in using the default address for the location.

Recommendation: Consider using see `transient`, for example in the following way:

```
bool transient isDisbursing_;
```

Optimism: Acknowledged. Won't apply since this means introducing a new compiler version on the monorepo that hasn't been used yet. This can be applied if in the future compiler versions are unified to `>= 0.8.28`.

Cantina Managed: Acknowledged.

3.3.10 `@custom:legacy`

Severity: Informational

Context: FeeVault.sol#L42-L47, FeeVault.sol#L170-L193

Description: The tag `@custom:legacy` is used for deprecated code. The event `Withdrawal` is also deprecated but doesn't have the tag.

Recommendation: Consider adding the tag `@custom:legacy` to the event `Withdrawal`, to make the use of this tag consistent.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.11 `totalProcessed` could be gamed

Severity: Informational

Context: FeeVault.sol#L146-L168

Description: For all the vaults, the `recipient` could game the value of `totalProcessed`. This could be done by first sending ETH to the vault and then retrieving it. This could also be done in a flash loan to reach very high numbers. In combination with the `FeeSplitter` this could also happen, depending on the results of the `SharesCalculator`.

Recommendation: Make sure the `recipient` is fully trusted when `totalProcessed` is used and relied on.

Optimism: Acknowledged. The value is used, but `recipient` is trusted.

Cantina Managed: Acknowledged.

3.3.12 `Feevault` specification is different than implementation

Severity: Informational

Context: FeeVault.sol#L31-L37, FeeVault.sol#L174-L193

Description: The Feevault specification indicates that for the functions `recipient()`, `minWithdrawalAmount()` and `withdrawalNetwork()`, the following must be done:

- Return the storage-configured xyz if a storage override has been set via setXyz.

- Otherwise return the legacy immutable xyz value.

However this is not present in the implementation. Possibly the specification is outdated.

Recommendation: Doublecheck and update the specification.

Optimism: Fixed in commit b81d1cf4.

Cantina Managed: Fix verified.

3.3.13 `L1Withdrawer` specification is different than implementation

Severity: Informational

Context: L1Withdrawer.sol#L69-L80

Description: The L1Withdrawer specification suggests using `L2ToL1MessagePasser.initiateWithdrawal()`. However the implemenation uses `sendMessage()`. The use of `sendMessage()` is a better solution because it allows failed withdrawal messages to be replayed.

Recommendation: Consider updating the specification.

Optimism: Fixed in commit b81d1cf4.

Cantina Managed: Fix verified.

3.3.14 `@notice` Semantic version not everywhere

Severity: Informational

Context: FeeSplitter.sol#L65-L66, L1Withdrawer.sol#L53-L55, SuperchainRevSharesCalculator.sol#L33-L34

Description: Most contracs have the following version information:

```
/// @notice Semantic version.
/// @custom:semver 1.0.0
string public constant version = "1.0.0";
```

However in the contracts `FeeSplitter` and `SuperchainRevSharesCalculator`, the `@notice Semantic version.` is missing. This is inconsistent.

Recommendation: Consider adding the `@notice Semantic version.` everywhere.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.15 Calibration of `feeDisbursementInterval`

Severity: Informational

Context: SuperchainRevSharesCalculator.sol#L66, SuperchainRevSharesCalculator.sol#L82-L95

Description: The fee calculation is based on two formulas, where the max is taken of the two. This means there will be a kink in the function. So the result of the formula can change based on when the underlying vaults are filled in relation to the calls to `disburseFees()`.

Recommendation: Check all the vaults for a period of one day (the default value of `feeDisbursementInterval`). Check all (major) additions are at least done one during that period. If there are additions that are done with a lower frequency, then consider increasing the `feeDisbursementInterval`.

Optimism: Acknowledged. We can add this recommendation in the documentation for the chain operators.

Cantina Managed: Acknowledged.

3.3.16 Inaccurate comment in `SuperchainRevSharesCalculator`

Severity: Informational

Context: SuperchainRevSharesCalculator.sol#L15

Description: The comment about `FeeSplitter's remainder send` isn't accurate because `FeeSplitter` doesn't have the concept of a remainder. The values are entirely determined by the SharesCalculator.

Recommendation: Consider making the comment more general.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.17 Incorrect location of vaults in `Predeploys.t.sol`

Severity: Informational

Context: FeeVault.sol#L84-L95, Predeploys.t.sol#L31-L43

Description: The vaults `SEQUENCER_FEE_WALLET`, `BASE_FEE_VAULT`, `L1_FEE_VAULT` and `OPERATOR_FEE_VAULT` are initializable because `FeeVault` contains the function `initialize()` and all vaults inherit from that. This means they are referenced in the wrong function in `Predeploys.t.sol`.

Recommendation: Consider moving the vault from `_usesImmutables()` to `_isInitializable()`.

Optimism: Fixed in PR 646.

Cantina Managed: Fix verified.

3.3.18 Different interfaces for `getRecipientsAndAmounts()`

Severity: Informational

Context: ISharesCalculator.sol#L24-L29, ISuperchainRevSharesCalculator.sol#L20-L25

Description: The similar functions `ISharesCalculator::getRecipientsAndAmounts()` and `ISuperchain-RevSharesCalculator::getRecipientsAndAmounts()` have slightly different parameter names. This could be confusing.

Recommendation: Consider using the same parameter names.

Optimism: Fixed in PR 641.

Cantina Managed: Fix verified.

3.3.19 `devnet-sdk/contracts/constants/constants.go` are missing the `OperatorFeeVault` and `FeeSplitter` addresses

Severity: Informational

Context: constants.go#L8-L46

Description: `devnet-sdk/contracts/constants/constants.go` are missing the `OperatorFeeVault` and `FeeSplitter` addresses.

Recommendation: Although they don't have a use and aren't referenced anywhere now, they can be added now for future use if needed.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.20 `ChainFeesRecipientRole` can be added as a devkey

Severity: Informational

Context: devkeys.go#L137-L194

Description: `ChainFeesRecipientRole` can be added as a devkey in `op-chain-ops/devkeys/devkeys.go`.

Recommendation: While not used or referenced anywhere, it can be added here now for future use if needed.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.21 Different solidity versions

Severity: Informational

Context: FeesDepositor.sol#L2, FeeSplitter.sol#L2, FeeVault.sol#L2

Description: `FeesDepositor` uses solidity `0.8.15`, while the other contracts use `0.8.25`.

Note: `FeesDepositor` is for `L1` and the other contracts are for `L2`.

Recommendation: Consider using the same solidity version everywhere.

Optimism: Acknowledged. `FeesDepositor` inherits from `ReinitializableBase`, that's why we need the `0.8.15` version.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3.22 `Require()` with string based error message

Severity: Informational

Context: IFeeVault.sol#L7-L9, FeeVault.sol#L146-L160

Description: Function `withdraw()` uses strings as errors. In most other locations, custom errors are used.

Recommendation: Consider using custom errors everywhere.

Optimism: Acknowledged. This is out of scope for this project and is no security issue.

Cantina Managed: Acknowledged.

3.3.23 `testVaultsWithoutRevenueShare()` doesn't have `operatorFeeVault`

Severity: Informational

Context: L2Genesis.t.sol#L68-L71, L2Genesis.t.sol#L86-L90

Description: The test `testVaultsWithoutRevenueShare()` doesn't have `operatorFeeVault`, while the related test `testVaultsWithRevenueShare()` does have this.

Recommendation: Consider also adding `operatorFeeVault` to `testVaultsWithoutRevenueShare()`.

Optimism: Fixed in PR 646.

Cantina Managed: Fix verified.

3.3.24 `OperatorFeeVault` can be included in `TestWithdrawalNetworkInlineJSON` test

Severity: Informational

Context: withdrawal_network_test.go#L80-L86

Description: `TestWithdrawalNetworkInlineJSON` test in `op-chain-ops/genesis/withdrawal_network_test.go` can include `OperatorFeeVault`.

Recommendation: `OperatorFeeVault` can be included in `TestWithdrawalNetworkInlineJSON` test.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.25 `FeeVaultWithdrawal.s.sol` no longer exists

Severity: Informational

Context: style-guide.md#L394-L397

Description: The `style-guide` mentions `FeeVaultWithdrawal.s.sol`, however this no longer exists. The last version that contained it was: v1.9.0

Recommendation: Consider updating the documentation.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.26 Documentation `init.md` doesn't contain `operatorFeeVaultRecipient`

Severity: Informational

Context: init.md#L42-L46, init.md#L63-L70

Description: The documentation `init.md` references three vault recipients, but `operatorFeeVaultRecipient` is missing. Also the list of multisigs might not be complete with regards to the `FeeSplitter`.

Recommendation: Consider adding `operatorFeeVaultRecipient` and doublecheck the list of multisigs.

Optimism: Fixed in PR 650.

Cantina Managed: Fix verified.

3.3.27 `operatorFeeVaultRecipient` could be added to `op-deployer/pkg/deployer/integration_test/apply_test.go`

Severity: Informational

Context: apply_test.go#L460-L502

Description: "operator fee vault recipient not set" test could be added to `op-deployer/pkg/deployer/integration_test/apply_test.go`.

Recommendation: Add additional test to `op-deployer/pkg/deployer/integration_test/apply_test.go`.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.28 Missing `OperatorFeeVault` and `FeeSplitter` bindings in `op-e2e/bindings`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Missing `OperatorFeeVault` and `FeeSplitter` bindings in `op-e2e/bindings`.

Recommendation: Bindings can be autogenerated. See the PR 17761. PR must be merged into the local repo first.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.

3.3.29 `l2GenesisOverrides` struct does not need to include `useRevenueShare`

Severity: Informational

Context: l2genesis.go#L24-L37, l2genesis.go#L166-L181

Description: `l2GenesisOverrides` struct does not need to include `useRevenueShare`. `useRevenueShare` is already included in the individual chain intent. When running the `L2Genesis.s.sol` script it uses the individual chain intent `thisIntent`.

```
if err := script.Run(opcm.L2GenesisInput{
    // ...
    UseRevenueShare: thisIntent.UseRevenueShare,
    // ...
}); err != nil {
    return fmt.Errorf("failed to call L2Genesis script: %w", err)
}
```

Including it in `l2GenesisOverrides` implies the a global or per-chain override can override the value specified by the intent which isn't true.

Recommendation: Remove `useRevenueShare` from `l2GenesisOverrides`.

Optimism: Fixed in PR 639.

Cantina Managed: Fix verified.