Final Commit: 89f0e17834003de24d6d49da70dd10bac26f40b6

Low Issues

L-01. Incorrect rateLimiter consumption for full exits

Description: The following formula is used in getTotalEthRequested() to determine the totalGwei:

```javascript
uint256 gweiAmount = requests[i].amountGwei == 0 ? FULL_EXIT_GWEI :
uint256(requests[i].amountGwei);
```

However, this incorrectly assumes that every full exit is withdrawing 2_048_000_000_000.

As a result, full withdrawals will overconsume an incorrect amount of gwei for full exits.

Recommendation: Consider supplying only the exact amounts during a withdrawal for an accurate consumption.

Customer's response: This is a known assumption and a non-blocker, after we finish consolidating this will be much more close to accurate. Also Eigenpod does not track individual validator balance on the execution layer so there is no way to fetch them. They only sum the last known balances when doing a checkpoint proof.

Fix Review: Acknowledged

L-02. Some withdrawals may not be processed, leading to overconsumption of the rateLimiter and loss of fees

Description: In the eigenPod the following conditions are not explicitly validated, but may lead to a failed processing of the withdrawal request:

```JavaScript
   /// Some requirements that are NOT checked by the pod:
     /// - request.pubkey MUST be a valid validator pubkey
    /// - request.pubkey MUST belong to a validator whose withdrawal
credentials are this pod
    /// - If request.amount is for a partial exit, the validator MUST have 0x02
withdrawal credentials
    /// - If request.amount is for a full exit, the validator MUST NOT have any
pending partial exits
    /// - The validator MUST be active and MUST NOT have initiated exit
```

As a result, if a withdrawal request is sent, but does not satisfy any of these conditions, it will not be processed, but still consumed from the rateLimiter and send the msg.value.

Recommendation: Consider implementing additional validation, to ensure all requests are processed.

Customer's response: This is also a non-blocker, since this will be our fault if it happens and the admin of the EtherFiRateLimiter could always fix it if we screwed up. Also there is no way we can know if it failed on the beacon layer from the execution layer because we cannot query that state. It is up to us to call this method correctly

Fix Review: Acknowledged

L-03. Fee overpayment will not be returned to the caller of requestConsolidation and requestExecutionLayerTriggeredWithdrawal

Description: In the eigenPod, any fee overpayment is returned to the msg.sender:

```JavaScript
  // Refund remainder of msg.value
        if (remainder > 0) {
            Address.sendValue(payable(msg.sender), remainder);
        }
```

The eigenPod::requestWithdrawal() and eigenPod::requestConsolidation() are invoked by the EtherFiNode contract, so it is going to be the receiver of the remainder. However, in the sweepFunds() function, any balance that is held in the Node is transferred to the liquidityPool:

```JavaScript
   function sweepFunds() external onlyEtherFiNodesManager returns (uint256
balance) {
        uint256 balance = address(this).balance;
        if (balance > 0) {
            (bool sent,) = payable(address(liquidityPool)).call{value: balance,
gas: 20000}("");
            if (!sent) revert TransferFailed();
            emit FundsTransferred(address(liquidityPool), balance);
        }
        return balance;
    }
```

As a result, any fee overpayment will be sent to the liquidityPool instead of the original payer of the fee.
Recommendation: Consider returning the fee to the payer, instead of the liquidity pool.

Customer's response: This is again known and a non-blocker. If this happens it is our fault and the sum (currently 1 gwei/request) accrued in liquidityPool and can be claimed by the admin eventually.

Fix Review: Acknowledged

L-04. setRemaining on bucketLimiter does not cap to capacity

Description: The setRemaining() function in EtherFiRateLimiter:

```JavaScript
function setRemaining(bytes32 id, uint64 remaining) external onlyAdmin {
        if (!limitExists(id)) revert UnknownLimit();

        BucketLimiter.setRemaining(limits[id], remaining);
        emit RemainingUpdated(id, remaining);
    }
```

Uses the underlying BucketLimiter library which defines setRemaining like this:

```JavaScript
function setRemaining(Limit storage limit, uint64 remaining) internal {
        refill(limit);
        limit.remaining = remaining;
    }
```

The issue is that the remaining field is set directly to the new value, but there is no check on the capacity as in all other updating functions in the library. This allows to set remaining to a value which is beyond the max capacity, which should not be allowed

Recommendation: Cap the remaining value up to the size of capacity

```JavaScript
function setRemaining(Limit storage limit, uint64 remaining) internal {
        refill(limit);
        limit.remaining = remaining > limit.capacity ? limit.capacity :
remaining ;
    }
```

Customer's response: This is a valid issue, thank you for pointing it out. I've fixed it in my next commit. I've added the line in the library itself.

Fix Review: Fixed in commit f50b87

L-05. Invalid consolidation requests can be made, leading to loss of fees

Description: The following conditions are required for the correct processing of a consolidation request but are not checked by the eigenPod:

```JavaScript
      /// Some requirements that are NOT checked by the pod:
    /// - If request.srcPubkey == request.targetPubkey, the validator MUST have
0x01 credentials
    /// - If request.srcPubkey != request.targetPubkey, the target validator
MUST have 0x02 credentials
    /// - Both the source and target validators MUST be active on the beacon
chain and MUST NOT have
    ///   initiated exits
    /// - The source validator MUST NOT have pending partial withdrawal
requests (via `requestWithdrawal`)
    /// - If the source validator is slashed after requesting consolidation
(but before processing),
    ///   the consolidation will be skipped.
```

As a result, if a request does not satisfy any of these requirements, the processing of that request will be failed, but the fee will still be paid.

For example the function assumes that *"eigenlayer will revert if all validators don't belong to the same pod"* and uses the srcPubkey of the first request to get the eigenPod node. However in eigenlayer only the targetPubKey is verified against the Pod. This means that requestConsolidation() can be called with source validator keys that do not belong to the Pod and eigenlayer will still **not** revert, making the transaction succeed. This will subsequently fail silently on the beacon chain

Recommendation: Consider adding additional validation in order to avoid sending invalid requests.

Customer's response: This probably might be an issue but since only Admin can call this function, this will be our fault if it happens, thus a non-blocker.

Fix Review: Acknowledged

L-06.Wrong target address defined in DiscoverCurrentWhitelist script

Description: Inside the getAllTargets() function of the script the LiquidityPool address is wrong. It is set to 0x35F4f28A8d3Ff20EEd10e087e8F96Ea2641E6AA2 (which is the contract of another protocol), while it should be 0x308861A430be4cce5502d0A12724771Fc6DaF216 as pointed in the deployed addresses in the docs

Additionally consider:

- Reducing the targets array size from 25 to 8 to reduce the unnecessary loops
- Inside CleanupOldWhitelist script verifyCleanup() is never run at the end of run()

Recommendation: Address the above inconsistencies

Customer's response: Fixed the liquidity Pool address and targets size array issue. The `verifycleanup()` is supposed to be called explicitly for this script. There is a bigger deployment and verification script, that does the actual cleanup, deployment and upgrade which is this.

Fix Review: Fixed in 581c602

Informational Issues

I-01.Revert on zero address when calling updateConsumer()

Description: Inside updateConsumers() function of EtherFiRateLimiter, add sanity check that prevents address(0) to be enabled as consumer

Recommendation: Add sanity check

Customer's response: Acknowledged

Fix Review: Acknowledged