Final Commit: 037da63f453b943e7bd96c155e0798003094e4a0

H-01. Redemptions with stEth will cause reward dilution and permanent locking of rewards

Description: In both the redeemEEthWithPermit() and redeemWeEthWithPermit() the output token can be set to Eth or stEth. When stEth is used, the user will transfer in eEth/weEth and the restaker is going to release stEth:

```JavaScript
IERC20(address(eEth)).safeTransferFrom(msg.sender, address(this), eEthAmount);
...
etherFiRestaker.transferStETH(receiver, eEthAmountToReceiver);
```

This approach locks eEth into the EtherFiRedemptionManager, which will still earn rewards (to the detriment of other stakers), but must **NEVER** be redeemed, as the tokens which were backing it were already transferred to the receiver.

Recommendation: Consider using an alternative approach in order to minimize discrepancies between the Eth and stEth outputs and avoid any stuck tokens. For example:
1) User transfers in the eEth amount

2) eEth is burned from the contract.

3) liquidityPool::rebase() is invoked, which will decrease the totalValueOutOfLp by the stEth amount that the user is going to receive (requires updating the rebase() function to also reduce the value of totalValueOutOfLp)

4) Invoke transferStEth

Customer's response: Fixed in commit 4151ea3d

Fix Review: Fixed

Medium severity findings

M-01.Approvals are spend, without being used

Description: In both the redeemEEthWithPermit() and redeemWeEthWithPermit() the receiver is passed in as the owner field of the permit function:

```
JavaScript
try eEth.permit(receiver, address(this), permit.value, permit.deadline,
permit.v, permit.r, permit.s) {}
```

However, eEth is always transferred from msg.sender. As a result, anytime the receiver is not the msg.sender, the functions will consume the permit of the receiver, but it will try to transfer from msg.sender and revert. Or in case msg.sender has given enough approval prior to that it will consume the receiver permit, without actually utilizing it

Recommendation: Consider replacing back the receiver with msg.sender.

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

Low severity findings

L-01. Low watermark threshold not validated properly

Description: The redemption manager implements a watermark check when calling canRedeem() to make sure the balances are not below the expected minimum in order for instant redemptions to be allowed.

```
JavaScript
function canRedeem(uint256 amount, address token) public view returns (bool) {
        uint256 liquidEthAmount = getInstantLiquidityAmount(token);
        if (liquidEthAmount < lowWatermarkInETH(token)) {
            return false;
        }
        uint64 bucketUnit = _convertToBucketUnit(amount, Math.Rounding.Up);
        bool consumable =
BucketLimiter.canConsume(tokenToRedemptionInfo[token].limit, bucketUnit);
        return consumable && amount <= liquidEthAmount;
    }
```

The check is not complete and allows the balances to still be reduced below the watermark after a redemption. Consider the following example:

- Balance in redemption manager is 100 tokens and watermark is 50
- Bob wants to instantly redeem 70 tokens
- Since liquidEthAmount > lowWatermarkInETH(token) it would be allowed
- Bob withdraws 70 tokens and the balances left in the manager are 30, which is below the 50 watermark minimum

Recommendations: To proper way to check for how much can be redeemed, without affecting the minimum watermark, is to check for the difference between liquidEthAmount & lowWatermarkInETH(token) and only then allow a redeem up to that balance:

```
JavaScript
 if (liquidEthAmount <= lowWatermarkInETH(token)) {
```

```
            return false;
  }

availableToRedeem = liquidEthAmount - lowWatermarkInETH(token)

if (amount > availableToRedeem) {return false}
```

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

L-02 canRedeem will return false, even if redemptions are possible

Description: The canRedeem() function will perform all the validations based on the amount of eEth or weEth. However, this will include the amount from the feeShareToStakers and the eEthFeeAmountToTreasury, which do not require any available stEth tokens.

```JavaScript
  require(canRedeem(eEthAmount, outputToken), "EtherFiRedemptionManager:
Exceeded total redeemable amount");
```

As a result, even if there are enough stEth tokens to satisfy the withdrawal, the canRedeem function will still return false.

Recommendations: Consider performing the canRedeem check only on the eEthAmountToTransfer.

Customer's response: *"The fee is very small should not make significant difference and allows us to keep the implementation the same per token and explaining the calculation is easier"*

Fix Review: Acknowledged

Informational findings

I-01. Comments are not updated

Description: In the EtherFiRedemptionManager, NatSpec and comments do not mention anything regarding the new StEth logic.

Recommendations: Consider updating the comments.

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

I-02. Additional sanity check

Description: Upon calling _processETHRedemption() there is the following sanity check:

```javascript
require(liquidityPool.withdraw(address(this), eEthAmountToReceiver) ==
sharesToBurn, "invalid num shares burnt");
```

It makes sure that the amount of shares sent is relevant to the amount of assets received. The returned shares of liquidityPool.withdraw() is the result of calling sharesForWithdrawalAmount()

The same sanity check should also be added to _processStETHRedemption(), where even though there are no actual shares being burned, there should still be a check to make sure that the eEthAmountToReceiver being sent as stETH equals to the amount of shares being locked into the redemption manager.

Recommendations: Call sharesForWithdrawalAmount(eEthAmountToReceiver) and do a sanity check before transferring stETH

Customer's response: Acknowledged

Fix Review: Acknowledged

I-03 Users will earn slightly more by splitting their redemptions

Description: The _processStETHRedemption will pay a fee to the stakers, by burning a portion of the shares without transferring out any liquidity. As a result, upon each redemption, the share price will be slightly higher:

```JavaScript
liquidityPool.burnEEthShares(feeShareToStakers);
```

As a result, users will be able to earn slightly more tokens, by splitting the redemptions.

Recommendations: This is part of the redemption design and it is important that the team is aware of this side effect

Customer's response: Acknowledged

Fix Review: Acknowledged

I-04 Attackers may delay normal withdrawals

Description:
The following flow will be used for normal withdrawals from the stEth strategy:
1) EtherFiRestaker::queueWithdrawals()

2)EtherFiRestaker::completeQueuedWithdrawals()

3) EtherFiRestaker::stEthRequestWithdrawal()

4) EtherFiRestaker::stEthClaimWithdrawals()

However, anyone will be able to disrupt this flow, by redeeming stEth between steps 2 and 3, potentially delaying the normal withdrawal flow.

Recommendations: Consider executing steps 2 and 3 in the same transaction to avoid such issues.

Customer's response: *"We will perform 2 and 3 in 1 tx if we need to perform this flow"*

Fix Review: Acknowledged