SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: March 20 - March 23, 2023
Prepared on: April 6, 2023

Introduction

Olympus is building OHM, a community-owned, decentralized and censorship-resistant reserve currency that is asset-backed, deeply liquid and used widely across Web3.

Scope

sherlock-olympus @ e502fe566516f358141118a40f1c02e014f8b27c

- sherlock-olympus/src/policies/BoostedLiquidity/BLVaultLido.sol
- sherlock-olympus/src/policies/BoostedLiquidity/BLVaultManagerLido.sol
- sherlock-olympus/src/policies/BoostedLiquidity/interfaces/IBLVaultLido.sol
- sherlock-olympus/src/policies/BoostedLiquidity/interfaces/IBLVaultManagerLido.sol

The in-scope contracts depend on these previously audited and external contracts:

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
2	5

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
cducrest-brainbot
RaymondFam
hickuphh3
Bahurum
carrot
chaduke

Issue H-1: Users can abuse discrepancies between oracle and true asset price to mint more OHM than needed and profit from it

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/5

Found by

0x52

Summary

All chainlink oracles have a deviation threshold between the current price of the asset and the on-chain price for that asset. The more oracles used for determining the price the larger the total discrepancy can be. These can be combined and exploited to mint more OHM than expected and profit.

Vulnerability Detail

BLVaultLido.sol#L156-L171

```
uint256 ohmWstethPrice = manager.getOhmTknPrice();
uint256 ohmMintAmount = (amount_ * ohmWstethPrice) / _WSTETH_DECIMALS;

// Block scope to avoid stack too deep
{
    // Cache OHM-wstETH BPT before
    uint256 bptBefore = liquidityPool.balanceOf(address(this));

    // Transfer in wstETH
    wsteth.safeTransferFrom(msg.sender, address(this), amount_);

    // Mint OHM
    manager.mintOhmToVault(ohmMintAmount);

    // Join Balancer pool
    _joinBalancerPool(ohmMintAmount, amount_, minLpAmount_);
```

The amount of OHM to mint and deposit is determined by the calculated price from the on-chain oracle prices.

BLVaultLido.sol#L355-L364

```
uint256[] memory maxAmountsIn = new uint256[](2);
maxAmountsIn[0] = ohmAmount_;
```


SHERLOCK

```
maxAmountsIn[1] = wstethAmount_;

JoinPoolRequest memory joinPoolRequest = JoinPoolRequest({
    assets: assets,
    maxAmountsIn: maxAmountsIn,
    userData: abi.encode(1, maxAmountsIn, minLpAmount_),
    fromInternalBalance: false
});
```

To make the issue worse, _joinBalancerPool use 1 for the join type. This is the EXACT_TOKENS_IN_FOR_BPT_OUT method of joining. What this means is that the join will guaranteed use all input tokens. If the current pool isn't balanced in the same way then the join request will effectively swap one token so that the input tokens match the current pool. Now if the ratio is off then too much OHM will be minted and will effectively traded for wstETH. This allows the user to withdraw at a profit once the oracle has been updated the discrepancy is gone.

Impact

Users can always time oracles so that they enter at an advantageous price and the deficit is paid by Olympus with minted OHM

Code Snippet

https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultLido.sol#L340-L370

Tool used

Manual Review

Recommendation

The vault needs to have withdraw and/or deposit fees to make attacks like this unprofitable.

Discussion

0xLienid

Similar underlying issues to #027 and #051. Solving one should solve all of them.

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/8/files

Issue H-2: Adversary can stake LP directly for the vault then withdraw to break lp accounting in BLVaultManager-Lido

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/4

Found by

carrot, Bahurum, hickuphh3, cducrest-brainbot, 0x52

Summary

The AuraRewardPool allows users to stake directly for other users. In this case the malicious user could stake LP directly for their vault then call withdraw on their vault. This would cause the LP tracking to break on BLVaultManagerLido. The result is that some users would now be permanently trapped because their vault would revert when trying to withdraw.

Vulnerability Detail

BaseRewardPool.sol#L196-L207

```
function stakeFor(address _for, uint256 _amount)
    public
    returns(bool)
{
    _processStake(_amount, _for);

    //take away from sender
    stakingToken.safeTransferFrom(msg.sender, address(this), _amount);
    emit Staked(_for, _amount);

    return true;
}
```

AuraRewardPool allows users to stake directly for another address with them receiving the staked tokens.

BLVaultLido.sol#L218-L224

```
manager.decreaseTotalLp(lpAmount_);

// Unstake from Aura
auraRewardPool().withdrawAndUnwrap(lpAmount_, claim_);
```


SHERLOCK

```
// Exit Balancer pool
_exitBalancerPool(lpAmount_, minTokenAmounts_);
```

Once the LP has been stake the adversary can immediately withdraw it from their vault. This calls decreaseTotalLP on BLVaultManagerLido which now permanently break the LP account.

BLVaultManagerLido.sol#L277-L280

```
function decreaseTotalLp(uint256 amount_) external override onlyWhileActive
↪   onlyVault {
    if (amount_ > totalLp) revert BLManagerLido_InvalidLpAmount();
    totalLp -= amount_;
}
```

If the amount_ is ever greater than totalLP it will cause decreaseTotalLP to revert. By withdrawing LP that was never deposited to a vault, it permanently breaks other users from being able to withdraw.

Example: User A deposits wstETH to their vault which yields 50 LP. User B creates a vault then stake 50 LP and withdraws it from his vault. The manager now thinks there is 0 LP in vaults. When User A tries to withdraw their LP it will revert when it calls manger.decreaseTotalLp. User A is now permanently trapped in the vault.

Impact

LP accounting is broken and users are permanently trapped.

Code Snippet

BLVaultLido.sol#L203-L256

Tool used

Manual Review

Recommendation

Individual vaults should track how much they have deposited and shouldn't be allowed to withdraw more than deposited.

Discussion

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/5/files

Issue H-3: minTokenAmounts_ is useless in new configuration and doesn't provide any real slippage protection

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/3

Found by

0x52

Summary

BLVaultLido#withdraw skims off extra stETH from the user that results from oracle arb. The problem with this is that minTokenAmounts_ no longer provides any slippage protection because it only ensures that enough is received from the liquidity pool but never enforces how much is received by the user.

Vulnerability Detail

BLVaultLido.sol#L224-L247

```
_exitBalancerPool(lpAmount_, minTokenAmounts_);

// Calculate OHM and wstETH amounts received
uint256 ohmAmountOut = ohm.balanceOf(address(this)) - ohmBefore;
uint256 wstethAmountOut = wsteth.balanceOf(address(this)) - wstethBefore;

// Calculate oracle expected wstETH received amount
// getTknOhmPrice returns the amount of wstETH per 1 OHM based on the oracle
↪  price
uint256 wstethOhmPrice = manager.getTknOhmPrice();
uint256 expectedWstethAmountOut = (ohmAmountOut * wstethOhmPrice) /
↪  _OHM_DECIMALS;

// Take any arbs relative to the oracle price for the Treasury and return the
↪  rest to the owner
uint256 wstethToReturn = wstethAmountOut > expectedWstethAmountOut
    ? expectedWstethAmountOut
    : wstethAmountOut;
if (wstethAmountOut > wstethToReturn)
    wsteth.safeTransfer(TRSRY(), wstethAmountOut - wstethToReturn);

// Burn OHM
ohm.increaseAllowance(MINTR(), ohmAmountOut);
manager.burnOhmFromVault(ohmAmountOut);
```


SHERLOCK

```
// Return wstETH to owner
wsteth.safeTransfer(msg.sender, wstethToReturn);
```

minTokenAmounts_ only applies to the removal of liquidity. Since wstETH is skimmed off to the treasury the user no longer has any way to protect themselves from slippage. As shown in my other submission, oracle slop can lead to loss of funds due to this skimming.

Impact

Users cannot protect themselves from oracle slop/wstETH skimming

Code Snippet

https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultLido.sol#L203-L256

Tool used

Manual Review

Recommendation

Allow the user to specify the amount of wstETH they receive AFTER the arb is skimmed.

Discussion

0xLienid

This is true, I'm not sure it would matter much in practice since there's no real economic incentive for anyone but the Treasury to do this. That said, it's a relatively easy fix so I think it's worth including anyways.

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/11/files

Issue H-4: stETH/ETH chainlink oracle has too long of heartbeat and deviation threshold which can cause loss of funds

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/2

Found by

0x52

Summary

getTknOhmPrice uses the stETH/ETH chainlink oracle to calculate the current price of the OHM token. This token valuation is used to determine the amount of stETH to skim from the user resulting from oracle arb. This is problematic since stETH/ETH has a 24 hour heartbeat and a 2% deviation threshold. This deviation in price could easily cause loss of funds to the user.

Vulnerability Detail

BLVaultManagerLido.sol#L458-L473

```
function getTknOhmPrice() public view override returns (uint256) {
    // Get stETH per wstETH (18 Decimals)
    uint256 stethPerWsteth = IWsteth(pairToken).stEthPerToken();

    // Get ETH per OHM (18 Decimals)
    uint256 ethPerOhm = _validatePrice(ohmEthPriceFeed.feed,
    ↪  ohmEthPriceFeed.updateThreshold);

    // Get stETH per ETH (18 Decimals)
    uint256 stethPerEth = _validatePrice(
        stethEthPriceFeed.feed,
        stethEthPriceFeed.updateThreshold
    );

    // Calculate wstETH per OHM (18 decimals)
    return (ethPerOhm * 1e36) / (stethPerWsteth * stethPerEth);
}
```

getTknOhmPrice uses the stETH/ETH oracle to determine the price which as stated above has a 24 hour hearbeat and 2% deviation threshold, this means that the price can move up to 2% or 24 hours before a price update is triggered. The result is that the on-chain price could be much different than the true stETH price.

BLVaultLido.sol#L232-L240

```
uint256 wstethOhmPrice = manager.getTknOhmPrice();
uint256 expectedWstethAmountOut = (ohmAmountOut * wstethOhmPrice) /
 ↪  _OHM_DECIMALS;

// Take any arbs relative to the oracle price for the Treasury and return the
 ↪  rest to the owner
uint256 wstethToReturn = wstethAmountOut > expectedWstethAmountOut
    ? expectedWstethAmountOut
    : wstethAmountOut;
if (wstethAmountOut > wstethToReturn)
    wsteth.safeTransfer(TRSRY(), wstethAmountOut - wstethToReturn);
```

This price is used when determining how much stETH to send back to the user. Since the oracle can be up to 2% different from the true price, the user can unfairly lose part of their funds.

Impact

User will be unfairly penalized due large variance between on-chain price and asset price

Code Snippet

https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultManagerLido.sol#L440-L455

https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultManagerLido.sol#L458-L473

Tool used

Manual Review

Recommendation

Use the stETH/USD oracle instead because it has a 1-hour heartbeat and a 1% deviation threshold.

Discussion

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/6/files

Issue H-5: Adversary can sandwich oracle updates to exploit vault

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/1

Found by

0x52

Summary

BLVaultLido added a mechanism to siphon off all wstETH obtained from mismatched pool and oracle prices. This was implemented to fix the problem that the vault could be manipulated to the attackers gain. This mitigation however does not fully address the issue and the same issue is still exploitable by sandwiching oracle update.

Vulnerability Detail

BLVaultLido.sol#L232-L240

```
uint256 wstethOhmPrice = manager.getTknOhmPrice();
uint256 expectedWstethAmountOut = (ohmAmountOut * wstethOhmPrice) /
 ↪  _OHM_DECIMALS;

// Take any arbs relative to the oracle price for the Treasury and return the
 ↪  rest to the owner
uint256 wstethToReturn = wstethAmountOut > expectedWstethAmountOut
    ? expectedWstethAmountOut
    : wstethAmountOut;
if (wstethAmountOut > wstethToReturn)
    wsteth.safeTransfer(TRSRY(), wstethAmountOut - wstethToReturn);
```

In the above lines we can see that the current oracle price is used to calculate the expected amount of wstETH to return to the user. In theory this should prevent the attack but an attacker can side step this sandwiching the oracle update.

Example:

The POC is very similar to before except now it's composed of two transactions sandwiching the oracle update. Chainlink oracles have a tolerance threshold of 0.5% before updating so we will use that as our example value. The current price is assumed to be 0.995 wstETH/OHM. The oracle price (which is about to be updated) is currently 1:1

```
Transaction 1:

Balances before attack (0.995:1)
Liquidity: 79.8 OHM 80.2 wstETH
Adversary: 20 wstETH

Swap OHM so that pool price matches pre-update oracle price:
Liquidity: 80 OHM 80 wstETH
Adversary: -0.2 OHM 20.2 wstETH

Balances after adversary has deposited to the pool:
Liquidity: 100 OHM 100 wstETH
Adversary: -0.2 OHM 0.2 wstETH

Balances after adversary sells wstETH for OHM (0.5% movement in price):
Liquidity: 99.748 OHM 100.252 wstETH
Adversary: 0.052 OHM -0.052 wstETH

Sandwiched Oracle Update:

Oracle updates price of wstETH to 0.995 OHM. Since the attacker already sold
↪  wstETH to balance
the pool to the post-update price they will be able to withdraw the full amount
↪  of wstETH.

Transaction 2:

Balances after adversary removes their liquidity:
Liquidity: 79.798 OHM 80.202 wstETH
Adversary: 0.052 OHM 19.998 wstETH

Balances after selling profited OHM:
Liquidity: 79.849 OHM 80.152 wstETH
Adversary: 20.05 wstETH
```

As shown above it's still profitable to exploit the vault by sandwiching the oracle updates. With each oracle update the pool can be repeatedly attacked causing large losses.

Impact

Vault will be attacked repeatedly for large losses

Code Snippet

https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultLido.sol#L203-L256

Tool used

Manual Review

Recommendation

To prevent this I would recommend locking the user into the vault for some minimum amount of time (i.e. 24 hours)

Discussion

0xLienid

This is kind of similar to #029 in terms of underlying cause (knowledge of oracle update price)

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/3/files

Issue M-1: SetLimit does not take into account burned OHM

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/48

Found by

cducrest-brainbot, chaduke

Summary

The function `setLimit()` may not be able to sufficiently restrict mint ability of manager.

Vulnerability Detail

The `setLimit()` function reverts when `newLimit_ < deployedOhm`, mintOhmToVault will revert if `deployedOhm + amount_ > ohmLimit + circulatingOhmBurned`. If the value of `circulatingOhmBurned` is high, and the admin can only set the limit above `deployedOhm`, they could end up in a state where they cannot limit the amount the vault is allowed to burn sufficiently. I.e. the vault is always able to mint at least `circulatingOhmBurned` new tokens.

Note that `circulatingOhmBurned` is never lowered (even when minting new tokens), so this value could grow arbitrarily high.

Impact

Lack of control of admin on mint ability of manager.

Code Snippet

SetLimit function:
https://github.com/sherlock-audit/2023-03-olympus/blob/main/sherlock-olympus/src/policies/BoostedLiquidity/BLVaultManagerLido.sol#L480-L483

Tool used

Manual Review

Recommendation

Use similar restrictions as in `mintOhmToVault()` for `setLimit` or lower `circulatingOhmBurned` when minting new OHM.

SHERLOCK

Discussion

0xLienid

Same issue as #018

0xLienid

Fix Implementation: https://github.com/0xLienid/sherlock-olympus/pull/2/files

Issue M-2: Normal users could be inadvertently grieved by the withdrawn ratios check

Source: https://github.com/sherlock-audit/2023-03-olympus-judging/issues/28

Found by

RaymondFam

Summary

The contract check on the withdrawn ratios of OHM and wstETH against the current oracle price could run into grieving naive users by taking any wstETH shifted imbalance as a fee to the treasury even though these users have not gamed the system.

Vulnerability Detail

Here is a typical scenario, assuming the pool has been initiated with total LP equal to sqrt(100_000 * 1_000) = 10_000. (Note: OHM: $15, wstETH: $1500 with the pool pricing match up with manager.getOhmTknPrice() or manager.getTknOhmPrice(), i.e. 100 OHM to 1 wstETH or 0.01 wstETH to 1 OHM. The pool token balances in each step below may be calculated via the Constant Product Simulation after each swap and stake.)

```
OHM token balance: 100_000
wstETH token balance: 1_000
Total LP: 10_000
```

1. A series of swap activities results in the pool shifted more of the LP into wstETH.

   OHM token balance: 90_909.1 wstETH token balance: 1_100 Total LP: 10_000

2. Bob calls deposit() by providing 11 wstETH where 1100 OHM is minted with 1100 - 90909.1 * 0.01 = 190.91 unused OHM burned. (Note: Bob successfully stakes with 909.09 OHM and 11 wstETH and proportionately receives 100 LP.)

   OHM token balance: 91_818.19 wstETH token balance: 1_111 Total LP: 10_100 User's LP: 100

3. Bob changes his mind instantly and proceeds to call withdraw() to remove all of his LP. He receives the originally staked 909.09 OHM and 11 wstETH. All OHM is burned but he is only entitled to receive 909.09 / 100 = 9.09 wstETH since the system takes any arbs relative to the oracle price for the Treasury and returns the rest to the owner.

OHM token balance: 90_909.1 wstETH token balance: 1_100 Total LP: 10_000 User's LP: 0

Impact

Bob suffers a loss of 11 - 9.09 = 1.91 wstETH (~ 17.36% loss), and the system is ready to trap the next user given the currently imbalanced pool still shifted more of the LP into wstETH.

Code Snippet

File: BLVaultLido.sol#L143-L200

File: BLVaultLido.sol#L203-L256

Tool used

Manual Review

Recommendation

Consider implementing a snapshot of the entry record of OHM and wstETH and compare that with the proportionate exit record. Slash only the differential for treasury solely on dissuading large attempts to shift the pool around, and in this case it should be 0 wstETH since the originally staked wstETH is no greater than expectedWstethAmountOut.

Discussion

0xLienid

True, but this will be very very minor in practice. It relies on assuming no arbitrage is ever taken, it will also be helped (but not eliminated) by the solution to #003