SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	Napier
Prepared by:	Sherlock
Lead Security Expert:	zzykxx
Dates Audited:	May 13 - May 20, 2024
Prepared on:	June 4, 2024

Introduction

Napier Finance is the liquidity hub for Yield Trading. This contest covers a router for Napier AMM and Curve TwoCrypto and modules to interact with various LSTs/LRTs.

Scope

Repository: napierfi/metapool-router

Branch: main

Commit: 213d967dc2bd525aaa832c7f87a6ac6dc5eafd1b

Repository: napierfi/napier-uups-adapters

Branch: main

Commit: f345d328c9d4fe1703853d05d6c10d226f9915a4

Repository: napierfi/napier-v1

Branch: main

Commit: 4962a3c4e8747fe7aee99096020dc7c649de091b

For the detailed scope, see the contest details.

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
9	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

zzykxx
Ironsidesec
merlin
blutorque
Drynooo
KupiaSec
no
whitehair0330
Varun_05
Bauer
fandonov
PNS
Bauchibred
BiasedMerc
ydlee
blackhole
karsar
yamato

Issue H-1: DOS in the claimWithdraw function due to an incorrect check of the lastFinalizedRequestId in the EEther-Adapter.sol

Source:
https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/55

Found by

Drynooo, KupiaSec, blutorque, merlin, whitehair0330

Summary

The `claimWithdrawal` function has an incorrect check of `lastFinalizedRequestId`, which lead to a DOS vulnerability in the `EEtherAdapter.claimWithdraw` function.

```
if (_requestId < ETHERFI_WITHDRAW_NFT.lastFinalizedRequestId()) revert
↪  RequestInQueue();
```

Vulnerability Detail

Let's discuss how `WithdrawRequestNFT` handles withdrawal request IDs. When `requestWithdraw` is called, the `nextRequestId` is increased by one.

```
uint256 requestId = nextRequestId++;
```

For a user to successfully call the `claimWithdraw` function, the admin of `WithdrawRequestNFT` must call `finalizeRequests` with our `requestId`:

```
function finalizeRequests(uint256 requestId) external onlyAdmin {
        lastFinalizedRequestId = uint32(requestId);
    }
```

So, if a `requestId` is created and the admin finalizes our request id, then the user will be able to claim the withdrawal amount.

However, the issue lies in the fact that `EEtherAdapter.claimWithdrawal` checks whether `_requestId >= ETHERFI_WITHDRAW_NFT.lastFinalizedRequestId()`, otherwise the call will fail.

```
if (_requestId < ETHERFI_WITHDRAW_NFT.lastFinalizedRequestId()) revert
↪  RequestInQueue();
```

However, when we examine the `WithdrawRequestNFT.claimWithdrawal` function, we see a completely different check:

```
require(tokenId < nextRequestId, "Request does not exist");
--> require(tokenId <= lastFinalizedRequestId, "Request is not finalized");
require(ownerOf(tokenId) != address(0), "Already Claimed");
```

From this, we can conclude that a user will only be able to call the `claimWithdrawal` function when `requestId = lastFinalizedRequestId`; otherwise, the call will fail.

Now, if we examine WithdrawRequestNFT on Etherscan, we can obtain the following information as of the report writing time:

Most importantly, the admin calls `lastFinalizedRequestId` for each request ID, and the user can claim the withdrawal request ID later, meaning the main condition is that `tokenId <= lastFinalizedRequestId`.

This will result in a situation where if the admin calls lastFinalizedRequestId with our request ID and then with the next one, we will never be able to claim the withdrawal by this request ID.

Impact

Users will not lose their shares or receive the expected ETH, but this will impact `EEtherAdapter` as a whole because the `totalQueueEth` will be increased by the requested withdrawal amount, and it will not be possible to decrease it due to the DOS of the `claimWithdraw` function. As `EEtherAdapter` is not an upgradable smart contract, I consider this issue to be of high severity.

Code Snippet

src/adapters/etherfi/EETHAdapter.sol#L63

Tool used

Manual Review

Recommendation

Consider removing this check altogether, or you can implement it exactly as `EtherFi` does:

```
-if (_requestId < ETHERFI_WITHDRAW_NFT.lastFinalizedRequestId()) revert
 ↪  RequestInQueue();
+if (_requestId > ETHERFI_WITHDRAW_NFT.lastFinalizedRequestId()) revert
 ↪  RequestInQueue();
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/napier-v1/pull/222

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-2: Users can frontrun LSTs/LRTs tokens prices decrease in order to avoid losses

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/65

The protocol has acknowledged this issue.

Found by

zzykxx

Summary

Users can redeem their `PT`/`YT` tokens before a price decrease of a supported LST/LRT token in order to avoid losses.

Vulnerability Detail

Napier allows users to redeem their `PT`/`YT` tokens for `ETH` via BaseLSTAdapter::prefundedRedeem() instantly if the amount to be withdrawn is lower or equal than the available `ETH` buffer. The in-scope adapters that allow this are:

- EETHAdapter
- UniEthAdapter

A Napier user that staked in one of these adapters can:

1. Monitor the mempool and the beacon chain to know in advance if either the `eETH` or `uniETH` tokens will lose value.
2. Frontrun the value loss by redeeming their `PT` and `YT`tokens via Tranche::redeemWithYT(), which will call BaseLSTAdapter::prefundedRedeem(), in exchange for `ETH`.

Because the value drop is still not reflected in the Napier protocol the staker will be able to withdraw his funds without being affected by the losses.

In the case of `eETH`, a rebase token, an attacker can know if a balance drop will happen by monitoring the mempool for calls to `rebase()` in the EtherFi LiquidityPool contract.

In the case of `uniEth` an attacker can know if the token will lose value by monitoring the protocol validators for penalties and slashing events. Bedrock (`uniEth`) is built on top of Eigenlayer, which can be notified of balance drops due to penalties or slashings via two permissionless functions: EigenPod::verifyBalanceUpdates() and

EigenPod::verifyAndProcessWithdrawals(). This allows an attacker to perform the following series of calls atomically to avoid losses:

1. Monitor the Bedrock validators on the beacon chain for penalties and slashings.
2. Call Tranche::redeemWithYT() to redeem `PT`/`YT` in exchange of `ETH`.
3. Call EigenPod::verifyBalanceUpdates()/EigenPod::verifyAndProcessWithdrawals() to notify Eigenlayer of the balance drop.
4. The value of `uniETH` will instantly drop.
5. Deposit the previously withdrawn `ETH` for more `YT`/`PT` tokens than the initial amount.

Another instance that instantly lowers the value held by the `UniEthAdapter` adapter is the call to UniETHAdapter::swapUniETHForETH() because a `0.05%` fee is paid to UniswapV3, this can also be front run by stakers to avoid bearing the losses of the fee.

Impact

Stakers can avoid losses, which implies honest stakers will lose more than they should.

Code Snippet

Tool used

Manual Review

Recommendation

Introduce a withdraw queue, this will prevent this kind of frontrunning attacks.

Discussion

massun-onibakuchi

It is known behavior seen in many LST/LRT integrations like DEX

Issue M-1: `currentStakeLimit` depletes faster in some adapters, due to actual amount spent less than the input `stakeAmount`

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/8

Found by

Varun_05, blutorque

Summary

Vulnerability Detail

In the `BaseLSTAdapterUpgradeable.prefundedDeposit()`, the stake amount is capped to the `currentStakeLimit`.This is to prevent the buffer from being completed drained.

```
// Update the stake limit state in the storage
$.packedStakeLimitData.setStorageStakeLimitStruct(data.updatePrevStakeLimit(curr
↪  entStakeLimit - stakeAmount));
```

Before the staking occur, its checks whether the `stakeAmount` exceed current stakeLimit, if not modify the new stake limit to `currentStakeLimit - stakeAmount`. The issue is, the actual amount going to be spent could possibly lower than the `stakeAmount`

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/BaseLSTAdapterUpgradeable.sol#L157-L158

```
// Actual amount of ETH spent may be less than the requested amount.
stakeAmount = _stake(stakeAmount); // stake amount can be 0
```

which means the stake limit that was updated previously does not account for the actual amount that we staked. I found one instance of adapters where this could possibly occur,

kelp/RsETHAdapter.sol: Input `stakeAmount` modified to lower value if its greater than the stakeLimit of RsETHDeposit pool,

Impact

With every `prefundedDeposit` call where excess WETH is left to stake, the stake limit will deplete faster.

SHERLOCK

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/BaseLSTAdapterUpgradeable.sol#L157

Tool used

Manual Review

Recommendation

The `_stake()` method do returns the actual spent amount, therefore I suggest to update the staking limit after the staking has been done.

```
/// INTERACT ///
// Deposit into the yield source
// Actual amount of ETH spent may be less than the requested amount.
stakeAmount = _stake(stakeAmount); // stake amount can be 0

/// WRITE ///
// Update the stake limit state in the storage
$.packedStakeLimitData.setStorageStakeLimitStruct(data.updatePrevStakeLimit(curr
↪  entStakeLimit - stakeAmount));
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/napierfi/napier-v1/pull/219 https://github.com/napierfi/napier-uups-adapters/pull/9

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-2: Depositing `stETH` to puffer finance will revert due to wrong implementation of `PufETHAdapter._stake` call

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/21

Found by

Bauer, Drynooo, Ironsidesec, KupiaSec, PNS, blackhole, blutorque, karsar, merlin, no, yamato, zzykxx

Summary

Reason: `PufETHAdapter._stake` will always revert due to wrong external call implementation. Impact: Can't deposit to Puffer. Likelihood: always.

Vulnerability Detail

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/puffer/PufETHAdapter.sol#L82

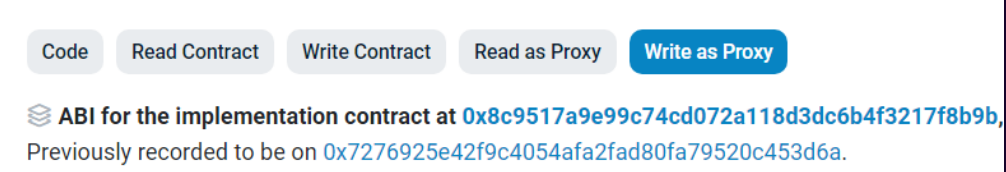
```
File: 2024-05-napier-update\napier-uups-adapters\src\adapters\puffer\PufETHAdapt
↪  er.sol

66:   function _stake(uint256 stakeAmount) internal override returns (uint256) {
...
74:
75:     IWETH9(Constants.WETH).withdraw(stakeAmount);
76:     uint256 _stETHAmt = STETH.balanceOf(address(this));
77:     STETH.submit{value: stakeAmount}(address(this));
78:     _stETHAmt = STETH.balanceOf(address(this)) - _stETHAmt;
79:     if (_stETHAmt == 0) revert InvariantViolation();
80:
81:     // Stake stETH to PufferDepositor
82:  >>> uint256 _pufETHAmt =
↪  PUFFER_DEPOSITOR.depositStETH(Permit(block.timestamp, _stETHAmt, 0, 0, 0));
84:
...
88:   }
```

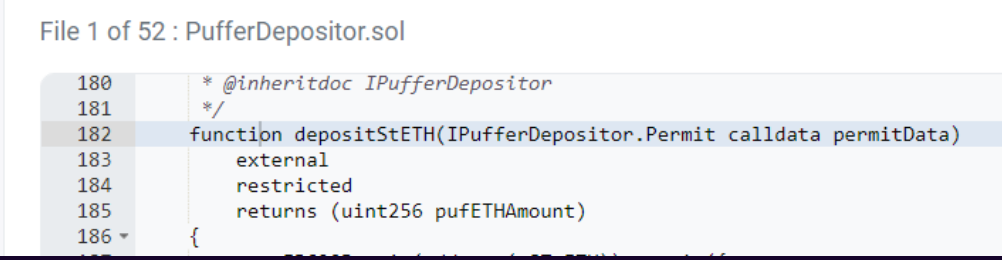
Issue flow:

1. When depositing by calling `PUFFER_DEPOSITOR.depositStETH(Permit)`, `PufETHAdapter` passes only one parameter `Permit` look at line 82 above.
2. But the current `PUFFER_DEPOSITOR.depositStETH` has 2 parameters (Permit, address recipient). Check https://etherscan.io/address/0x4aA799C5dfc01ee7d790e3bf1a7C2257CE1DcefF#writeProxyContract#F1.
3. This is due to the proxy upgrade of `PUFFER_DEPOSITOR` from implementation v1 to implementation v2.

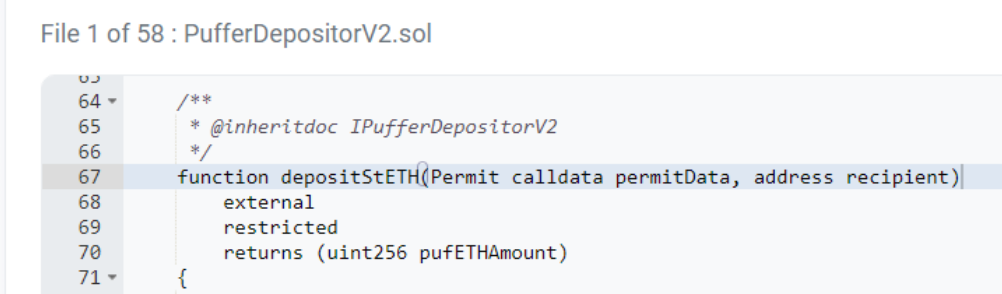
To check upgares of `PUFFER_DEPOSITOR`, scroll on https://etherscan.io/address/0x4aA799C5dfc01ee7d790e3bf1a7C2257CE1DcefF#writeProxyContract



Previous implementation where it had only one param https://etherscan.io/address/0x7276925e42f9c4054afa2fad80fa79520c453d6a#code#F1#L182



Latest implementation has 2 params https://etherscan.io/address/0x8c9517a9e99c74cd072a118d3dc6b4f3217f8b9b#code#F1#L67



Impact

Depositing stETH to puffer finance is not possible with current `PufETHAdapter`

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/puffer/PufETHAdapter.sol#L82

https://etherscan.io/address/0x8c9517a9e99c74cd072a118d3dc6b4f3217f8b9b#code#F1#L41

https://etherscan.io/address/0x4aA799C5dfc01ee7d790e3bf1a7C2257CE1DcefF#writeProxyContract

Tool used

Manual Review

Recommendation

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/puffer/PufETHAdapter.sol#L82

```
  function _stake(uint256 stakeAmount) internal override returns (uint256) {
...
    IWETH9(Constants.WETH).withdraw(stakeAmount);
    uint256 _stETHAmt = STETH.balanceOf(address(this));
    STETH.submit{value: stakeAmount}(address(this));
    _stETHAmt = STETH.balanceOf(address(this)) - _stETHAmt;
    if (_stETHAmt == 0) revert InvariantViolation();

    // Stake stETH to PufferDepositor
-    uint256 _pufETHAmt = PUFFER_DEPOSITOR.depositStETH(Permit(block.timestamp,
 ↪  _stETHAmt, 0, 0, 0));
+    uint256 _pufETHAmt = PUFFER_DEPOSITOR.depositStETH(Permit(block.timestamp,
 ↪  _stETHAmt, 0, 0, 0), address(this));

    if (_pufETHAmt == 0) revert InvariantViolation();

    return stakeAmount;
  }
```

Discussion

sherlock-admin2

SHERLOCK

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/napier-uups-adapters/pull/10

zzykxx

This has been fixed by changing the functionality of the `_stake()` function, now it deposits `WETH` instead of `stETH`.

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-3: Missing stake limit validation on `RenzoAdapter._stake`

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/24

Found by

Bauer, Ironsidesec, fandonov, zzykxx

Summary

Every `_stake` function in adapter contracts like `RsETHAdapter`, `PufETHAdapter`, and `RenzoAdapter` has the below @dev comment to implement the stake limit validation.

> /// @dev Need to check the current staking limit before staking to prevent DoS.

But only `RsETHAdapter`, `PufETHAdapter` validate the stake limits as shown below. But `RenzoAdapter` doesn't validate the stake limit and it reverts in an edge case.

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/puffer/PufETHAdapter.sol#L69-L73

```
File: 2024-05-napier-update\napier-uups-adapters\src\adapters\puffer\PufETHAdapter.sol

65: >>>  /// @dev Need to check the current staking limit before staking to prevent DoS.
66:   function _stake(uint256 stakeAmount) internal override returns (uint256) {
67:     if (stakeAmount == 0) return 0;
68:
69:     uint256 stakeLimit = STETH.getCurrentStakeLimit();
70:  >>>  if (stakeAmount > stakeLimit) {
71:       // Cap stake amount
72:       stakeAmount = stakeLimit;
73:     }
```

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L71-L75

```
File: 2024-05-napier-update\napier-uups-adapters\src\adapters\kelp\RsETHAdapter.sol
```


SHERLOCK

```
66: >>>  /// @dev Need to check the current staking limit before staking to
↪  prevent DoS.
67:   function _stake(uint256 stakeAmount) internal override returns (uint256) {
68:     if (stakeAmount == 0) return 0;
69:
70:     // Check LRTDepositPool stake limit
71:     uint256 stakeLimit =
↪  RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH);
72: >>>  if (stakeAmount > stakeLimit) {
73:       // Cap stake amount
74:       stakeAmount = stakeLimit;
75:     }
```

Vulnerability Detail

1. `RenzoAdapter._stake` calls `depositETH` on `RENZO_RESTAKE_MANAGER`

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/renzo/RenzoAdapter.sol#L59

```
File: 2024-05-napier-update\napier-uups-adapters\src\adapters\renzo\RenzoAdapter
↪  .sol

60: >>>  /// @dev Need to check the current staking limit before staking to
↪  prevent DoS.
61:   function _stake(uint256 stakeAmount) internal override returns (uint256) {
62:     if (stakeAmount == 0) return 0;
63:     if (RENZO_RESTAKE_MANAGER.paused()) revert ProtocolPaused();
64:     uint256 balanceBefore = EZETH.balanceOf(address(this));
65:     IWETH9(Constants.WETH).withdraw(stakeAmount);
66:     RENZO_RESTAKE_MANAGER.depositETH{value: stakeAmount}(0); //
↪  @audit-medium no referral id
67:     uint256 newBalance = EZETH.balanceOf(address(this));
68:     if (newBalance - balanceBefore == 0) revert InvariantViolation();
69:
70:     return stakeAmount;

70:   }
```

2. And look at `depositETH` line highlighted with >>> below, it checks the `MaxTVLReached`, and it will revert if max TVL is reached. Maybe someone

manipulated to cause DOS or unmanipulatedly hit the threshold triggering the revert. And the comment on `_stake` says to check the current limit to prevent DOS. But `RenzoAdapter._stake` is missing that.

https://etherscan.io/address/0xbaacd5f849024dcc80520baa952f11adfc59f9d0#code#F1#L558 Line 558 on https://etherscan.deth.net/address/0xbaacd5f849024dcc80520baa952f11adfc59f9d0

```
  function depositETH(uint256 _referralId) public payable nonReentrant notPaused
↪  {
    // Get the total TVL
    (, , uint256 totalTVL) = calculateTVLs();

    // Enforce TVL limit if set
>>>  if (maxDepositTVL != 0 && totalTVL + msg.value > maxDepositTVL) {
      revert MaxTVLReached();
    }

...
  }
```

Impact

DOS or Missing validation the dev intended to make but didn't implement.

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/renzo/RenzoAdapter.sol#L59

https://etherscan.io/address/0xbaacd5f849024dcc80520baa952f11adfc59f9d0#code#F1#L558

Line 558 on https://etherscan.deth.net/address/0xbaacd5f849024dcc80520baa952f11adfc59f9d0

Tool used

Manual Review

Recommendation

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/renzo/RenzoAdapter.sol#L59

```
  function _stake(uint256 stakeAmount) internal override returns (uint256) {
    if (stakeAmount == 0) return 0;
    if (RENZO_RESTAKE_MANAGER.paused()) revert ProtocolPaused();

+    uint maxDepositTVL = RENZO_RESTAKE_MANAGER.maxDepositTVL();
+    uint totalTVL = RENZO_RESTAKE_MANAGER.totalTVL();
+    if (maxDepositTVL != 0 && totalTVL + stakeAmount > maxDepositTVL) {
+      stakeAmount = maxDepositTVL - totalTVL;
+    }

    uint256 balanceBefore = EZETH.balanceOf(address(this));
    IWETH9(Constants.WETH).withdraw(stakeAmount);
    RENZO_RESTAKE_MANAGER.depositETH{value: stakeAmount}(0); // @audit-medium no
↪   referral id
    uint256 newBalance = EZETH.balanceOf(address(this));
    if (newBalance - balanceBefore == 0) revert InvariantViolation();

    return stakeAmount;
  }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/napier-uups-adapters/pull/11

zzykxx

An edge case in the proposed fix was found: `_stake()` reverts if `totalTVL` is greater than `maxDepositTVL`. It has been fixed in new PR:
https://github.com/napierfi/napier-uups-adapters/pull/22

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-4: Less rsETH minted than intended in volatile conditions. due to zero slippage when staking ETH to mint rsETH

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/26

The protocol has acknowledged this issue.

Found by

Bauchibred, BiasedMerc, Ironsidesec, merlin, zzykxx

Summary

This issue is not at all related to external admin, but the slippage parameter being zero is the issue. Read the issue flow after below images

Vulnerability Detail

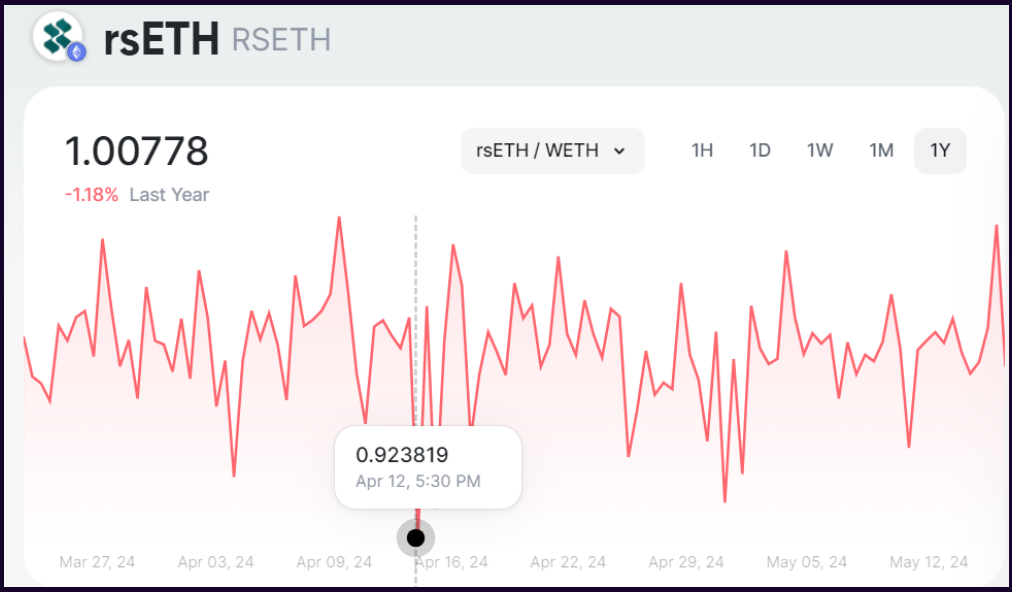
There is a slippage parameter called `minRSETHAmountExpected` in `RSETH_DEPOSIT_POOL.depositETH`. And `RsETHAdapter._stake` is setting it to 0 when it calls `RSETH_DEPOSIT_POOL.depositETH`.

Its an issue, look at the sudden waves / bumps of rsETH within seconds, and LRT oracle admin will update the price according to the markets in dexes, or else free arbitrage mints will further cause spiral moves.Or explore the ezETH depeg recently (https://x.com/SudipanSinha/status/1783473266515931284) and LRT x money market impact (https://x.com/SudipanSinha/status/1784107059744792715)

Click yellow button on https://www.dextools.io/app/en/ether/pair-explorer/0x059615ebf32c946aaab3d44491f78e4f8e97e1d3

https://app.uniswap.org/explore/pools/ethereum/0x059615EBf32C946aaab3D44491f78e4F8e97e1D3

https://matcha.xyz/tokens/ethereum/0xa1290d69c65a6fe4df752f95823fae25cb99e5a7?buyChain=1buyAddress=0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2



Issue flow:

1. Rebalancer notices ETH to deposit in Kelp DAO's rsETH.
2. call flows to `RsETHAdapter._stake` --> `RSETH_DEPOSIT_POOL.depositETH` which will deposit eth to mint rsETH at current price.
3. Its a volatile time, and many transactions are pending, kelp lrt oracle is updated according to latest dex price which has pumped a lot.
4. Then the napier's `_stake` transaction goes through, which will mint less rsETH because the price of rsETH went so high
5. but after few blocks, the price of rsETH again balanced and it dipped to normal, so lrt oracle is updated.

Now, it is a loss to Napier because we minted less tokens at peak price instead of normal current price and they are worth very low in terms of USD or WETH. If we used slippage, then it would have reverted in these cases.

Due to the below reason of price fluctuations, depegs, the slippage should not be set to 0.

And it is under the admin control. But during the times of rebalancing, if depeg happens or in a very volatile sessions, the slippage is necessary. Because the admin might update the latest rsETH/WETH price and we might receive less rsETH as intended.

This can be resulted to MEV or sandwich attacks to 0 slippage. Although rebalancers use private mempool, the price of rsETH will be volatile in big steps wise jumps as shown in images below. And the latest Renzo depeg of ezETH shows that slippage should be implemented, or else its a loss to Napier vaults.

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L84

```
File:
↪  2024-05-napier-update\napier-uups-adapters\src\adapters\kelp\RsETHAdapter.sol

67:     function _stake(uint256 stakeAmount) internal override returns (uint256)
↪  {
...
81:         // Interact
82:         IWETH9(Constants.WETH).withdraw(stakeAmount);
83:         uint256 _rsETHAmt = RSETH.balanceOf(address(this));
84:   >>>   RSETH_DEPOSIT_POOL.depositETH{value: stakeAmount}(0, REFERRAL_ID);
85:         _rsETHAmt = RSETH.balanceOf(address(this)) - _rsETHAmt;
86:
...
90:     }
```

https://etherscan.io/address/0x13576cd2b61e601d3e98b5c06ef81896c9bbb369#code#F1#L206 Line 206 on https://etherscan.deth.net/address/0x13576cd2b61e601d3e98b5c06ef81896c9bbb369

```
    function depositETH(
  >>>   uint256 minRSETHAmountExpected,
        string calldata referralId
    )
        external payable  whenNotPaused  nonReentrant
    {
        // checks
 >>>    uint256 rsethAmountToMint = _beforeDeposit(LRTConstants.ETH_TOKEN,
↪  msg.value, minRSETHAmountExpected);

        // interactions
        _mintRsETH(rsethAmountToMint);
        emit ETHDeposit(msg.sender, msg.value, rsethAmountToMint, referralId);
    }

    function _beforeDeposit(
        address asset,
        uint256 depositAmount,
 >>>    uint256 minRSETHAmountExpected
    )
        private view returns (uint256 rsethAmountToMint)
    {
```

```
... SKIP ...
        rsethAmountToMint = getRsETHAmountToMint(asset, depositAmount);

 >>>    if (rsethAmountToMint < minRSETHAmountExpected) {
            revert MinimumAmountToReceiveNotMet();
        }
    }

    function getRsETHAmountToMint(
        address asset,
        uint256 amount
    )
        public view  override   returns (uint256 rsethAmountToMint)
    {
        // setup oracle contract
        address lrtOracleAddress =
↪   lrtConfig.getContract(LRTConstants.LRT_ORACLE);
        ILRTOracle lrtOracle = ILRTOracle(lrtOracleAddress);

        // calculate rseth amount to mint based on asset amount and asset
↪   exchange rate
  >>>   rsethAmountToMint = (amount * lrtOracle.getAssetPrice(asset)) /
↪   lrtOracle.rsETHPrice();
    }
```

Impact

Less rsETH was minted than intended in volatile conditions.

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L84

Tool used

Manual Review

Recommendation

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L84

```
-   function _stake(uint256 stakeAmount) internal override returns (uint256) {
+   function _stake(uint256 stakeAmount, uint256 minRSETHAmountExpected)
 ↪  internal override returns (uint256) {

... SKIP ...

        // Interact
        IWETH9(Constants.WETH).withdraw(stakeAmount);
        uint256 _rsETHAmt = RSETH.balanceOf(address(this));
-       RSETH_DEPOSIT_POOL.depositETH{value: stakeAmount}(0, REFERRAL_ID);
+       RSETH_DEPOSIT_POOL.depositETH{value:
 ↪  stakeAmount}(minRSETHAmountExpected, REFERRAL_ID);
        _rsETHAmt = RSETH.balanceOf(address(this)) - _rsETHAmt;

        if (_rsETHAmt == 0) revert InvariantViolation();

        return stakeAmount;
    }
```

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

PNS commented:

> slippage control is (will be) at the Tranche.issue level and not at the adapter level (following issue #84 in the previous contest)

massun-onibakuchi

It makes sense but depegging upward is economically unlikely. some protocols seems to set zero slippage.

https://github.com/pendle-finance/pendle-core-v2-public/blob/77b3630c82412b580bce6cd4a32f2c385bbb7970/contracts/core/StandardizedYield/implementations/KelpDAO/PendleRsETHSY.sol#L79

TrancheRouter (TrancheRouter calls Tranche) checks how many PT a user should receive

amount of PT issued indirectly depends on rsETH price. so, I think TrancheRouter prevents such unexpected slippage. https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L107-L109 https://github.com/sherl

ock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/napier-uups-adapters/src/adapters/BaseLSTAdapterUpgradeable.sol#L87

Issue M-5: Slippage on `MetapoolRouter.addLiquidityOneETHKeepYt`

Source:
https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/28

Found by

Ironsidesec

Summary

As a user, I don't want to mint liquidity at a lower LP price and receive the manipulated / hugely changed YT price/amount.

A user can call `MetapoolRouter.addLiquidityOneETHKeepYt` to add liquidity (only PT) to metaPool (PT -tricrypto) and get YT (yield tokens) There's slippage protection `minLiquidity`, but no `minYT`. Due to volatility or MEV or normal events listed below, the amount of YT tokens received can be fewer which were issued at inflated price which dips within few blocks or receive huge at dipped price because there is no slippage to amount of YT you get.

Vulnerability Detail

1. Issue PT and YT using the received ETH
2. Add liquidity to the Curve metapool
3. Send the received LP token and YT to the recipient

Attack / issue flow :

1. The amount of liquidity to mint depends on the `3LST-PT TriCrypto LP token`, `Napier Principal Token`, and the global scales when issuing the PT + YT tokens.
2. So, if someone removed PT from twocrypto ,then adding even fewer PT tokens than normal will mint more liquidity. So, if the global scales or tranche issues changes, the value of PT decreases, so someone removes PT tokens from two crytpo LP, and since price of PT decreases, makes the amount of YT increase.
3. The exact opposite scenario can happen, where fewer YT tokes are issued but minimum liquidity slippage is passed. So, as a user I don't want to receive lower YT tokens when price of PT/YT suddenly changed. Add slippage to amount of PY issued.

During the issuance, the user will deposit underlying assets (e.g., ETH) to the Tranche contract, and the Tranche contract will forward them to the Adaptor contract for depositing. The number of shares minted is depending on the current

scale of the adaptor. The current scale of the adaptor can increase or decrease at any time, depending on the current on-chain condition when the transaction is executed. For instance, the LIDO's daily oracle/rebase update will increase the stETH balance, which will, in turn, increase the adaptor's scale. On the other hand, if there is a mass validator slashing event, the ETH claimed from the withdrawal queue will be less than expected, leading to a decrease in the adaptor's scale. Thus, one cannot ensure the result from the off-chain simulation will be the same as the on-chain execution.

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/metapool-router/src/MetapoolRouter.sol#L356-L392

```
File: 2024-05-napier-update\metapool-router\src\MetapoolRouter.sol

371:   function addLiquidityOneETHKeepYt(address metapool, uint256 minLiquidity,
↪  address recipient, uint256 deadline)
372:     external payable nonReentrant checkDeadline(deadline)
↪  checkMetapool(metapool) returns (uint256 liquidity)
378:   {
379:     // Steps:
380:     // 1. Issue PT and YT using the received ETH
381:     // 2. Add liquidity to the Curve metapool
382:     // 3. Send the received LP token and YT to the recipient
383:
...SNIP...
393:     uint256 pyAmount = pt.issue({to: address(this), underlyingAmount:
↪  msg.value});
395:
...SNIP...
401:     liquidity = Twocrypto(metapool).add_liquidity({
402:       amounts: [pyAmount, 0],
403:       min_mint_amount: minLiquidity,
404:       receiver: recipient
405:     });
406:
407:  >>>  IERC20(pt.yieldToken()).transfer(recipient, pyAmount);
409:   }
```

Impact

Loss of YT tokens. Unintended amount of YT is received.

SHERLOCK

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/metapool-router/src/MetapoolRouter.sol#L356-L392

Tool used

Manual Review

Recommendation

Implement a slippage control that allows the users to revert if the amount of YT they received is less than the amount they expected.

https://github.com/sherlock-audit/2024-05-napier-update/blob/c31af59c6399182fd04b40530d79d98632d2bfa7/metapool-router/src/MetapoolRouter.sol#L356-L392

```
-  function addLiquidityOneETHKeepYt(address metapool, uint256 minLiquidity,
 ↪  address recipient, uint256 deadline)
+  function addLiquidityOneETHKeepYt(address metapool, uint256 minLiquidity,
 ↪  uint256 minYT, address recipient, uint256 deadline)
    external
    payable
    nonReentrant
    checkDeadline(deadline)
    checkMetapool(metapool)
    returns (uint256 liquidity)
  {
...

    ITranche pt = ITranche(Twocrypto(metapool).coins(PEGGED_PT_INDEX));
    // Issue PT and YT using the received ETH
    if (_isApproved(address(WETH9), address(pt)) == 0) {
      _setApproval(address(WETH9), address(pt));
      WETH9.approve(address(pt), type(uint256).max);
    }
    uint256 pyAmount = pt.issue({to: address(this), underlyingAmount:
 ↪  msg.value});
+    if (pyAmount < minYT) revert InsufficientYTOut();

    // Add liquidity to the Curve metapool
    if (_isApproved(address(pt), metapool) == 0) {
      _setApproval(address(pt), metapool);
      pt.approve(metapool, type(uint256).max);
    }
```


SHERLOCK

```
...
  }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/metapool-router/pull/30

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-6: Incorrect checking in `receiveFlashLoan` can cause `swapETHForYt` to fail unexpectedly.

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/36

Found by

Drynooo, Ironsidesec, KupiaSec, Varun_05, no, ydlee

Summary

The following checking in `receiveFlashLoad` is incorrect, causing `swapETHForYt` to fail.

```
File: metapool-router/src/MetapoolRouter.sol

333:         if (repayAmount > remaining) revert
↪   Errors.MetapoolRouterInsufficientETHRepay(); // Can't repay the flash loan
```

Vulnerability Detail

In line 333, the check here is to ensure that the `ETH` sent by the user is enough to cover the `repayAmount`. Thus the check should revert if `repayAmount > TransientStorage.tloadU256(TSLOT_CB_DATA_VALUE)` , instead of `repayAmount > remaining`.

Let's assume that:

1. The user swaps `5 ETH` for some Yt by `swapETHForYt`. (i.e. `msg.value = 5 ETH`)
2. The estimated amount of WETH required to issue the PT and YT is `4 WETH` (i.e. `wethDeposit = 4 WETH`).
3. The PT are finally swapped to `2 WETH` (i.e. `wethReceived = 2 WETH`).
4. The `feeAmount` of flash loan is `0.1 WETH`.

Then:

1. `repayAmount = wethDeposit + feeAmounts[0] = 4 WETH + 0.1 WETH = 4.1 WETH`
2. `spent = repayAmount - wethReceived = 4.1 WETH - 2 WETH = 2.1 WETH`
3. `remaining = 5 WETH - spent = 5 WETH - 2.1 WETH = 2.9 WETH`

SHERLOCK

Line 333 in `receiveFlashLoan` reverts as `repayAmount > remaining`, and the `swapETHForYt` fails. But the user pays more than required to swap. The fail is not expected.

```
File: metapool-router/src/MetapoolRouter.sol

323:         // Calculate the amount of ETH spent in the swap
324:         uint256 repayAmount = wethDeposit + feeAmounts[0];
325:         uint256 spent = repayAmount - wethReceived; // wethDeposit +
 ↪  feeAmounts[0] - wethReceived
326:
327:         // Revert if the ETH spent exceeds the specified maximum
328:         if (spent > TransientStorage.tloadU256(TSLOT_CB_DATA_MAX_ETH_SPENT))
 ↪  {
329:             revert Errors.MetapoolRouterExceededLimitETHIn();
330:         }
331:
332:         uint256 remaining = TransientStorage.tloadU256(TSLOT_CB_DATA_VALUE)
 ↪  - spent;
333:@>       if (repayAmount > remaining) revert
 ↪  Errors.MetapoolRouterInsufficientETHRepay(); // Can't repay the flash loan
```

https://github.com/sherlock-audit/2024-05-napier-update/blob/main/metapool-router/src/MetapoolRouter.sol#L323-L333

Impact

Incorrect checking can cause `swapETHForYt` to fail unexpectedly, breaking the core functionality of the contract.

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/main/metapool-router/src/MetapoolRouter.sol#L323-L333

Tool used

Manual Review

Recommendation

```
-         if (repayAmount > remaining) revert
↪   Errors.MetapoolRouterInsufficientETHRepay();
+         if (repayAmount > TransientStorage.tloadU256(TSLOT_CB_DATA_VALUE))
↪   revert Errors.MetapoolRouterInsufficientETHRepay();
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/metapool-router/pull/28

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-7: Checking `RSETH_DEPOSIT_POOL.minAmountToDeposit()` in `RsETHAdapter::_stake()` causes Dos

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/46

The protocol has acknowledged this issue.

Found by

Drynooo, PNS, merlin, no, zzykxx

Summary

Checking `RSETH_DEPOSIT_POOL.minAmountToDeposit()` in `RsETHAdapter::_stake()` causes Dos

Vulnerability Detail

```
 function _stake(uint256 stakeAmount) internal override returns (uint256) {
        if (stakeAmount == 0) return 0;

        // Check LRTDepositPool stake limit
        uint256 stakeLimit =
↪   RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH);
        if (stakeAmount > stakeLimit) {
            // Cap stake amount
            stakeAmount = stakeLimit;
        }
        // Check LRTDepositPool minAmountToDeposit
@>        if (stakeAmount <= RSETH_DEPOSIT_POOL.minAmountToDeposit()) revert
↪   MinAmountToDepositError();
        // Check paused of LRTDepositPool
        if (RSETH_DEPOSIT_POOL.paused()) revert ProtocolPaused();

        // Interact
        IWETH9(Constants.WETH).withdraw(stakeAmount);
        uint256 _rsETHAmt = RSETH.balanceOf(address(this));
        RSETH_DEPOSIT_POOL.depositETH{value: stakeAmount}(0, REFERRAL_ID);
        _rsETHAmt = RSETH.balanceOf(address(this)) - _rsETHAmt;

        if (_rsETHAmt == 0) revert InvariantViolation();

        return stakeAmount;
```


SHERLOCK

```
    }
```

The _stake will revert in the condition that the stakeAmount is less than `RSETH_DEPOSIT_POOL.minAmountToDeposit()`, which is 100000000000000. This could always happens. Because stakeAmount is not the user's input, it is calculate by this protocal.

```
function prefundedDeposit() external nonReentrant onlyTranche returns (uint256,
↪  uint256) {
        LSTAdapterStorage storage $ = _getStorage();

        uint256 bufferEthCache = $.bufferEth; // cache storage reads
        uint256 queueEthCache = $.totalQueueEth; // cache storage reads
        uint256 assets = IWETH9(WETH).balanceOf(address(this)) - bufferEthCache;
↪  // amount of WETH deposited at this time
        uint256 shares = previewDeposit(assets);

        if (assets == 0) return (0, 0);
        if (shares == 0) revert ZeroShares();

        // Calculate the target buffer amount considering the user's deposit.
        // bufferRatio is defined as the ratio of ETH balance to the total
↪  assets in the adapter in ETH.
        // Formula:
        // desiredBufferRatio = (totalQueueEth + bufferEth + assets - s) /
↪  (totalQueueEth + bufferEth + stakedEth + assets)
        // Where:
        // assets := Amount of ETH the user is depositing
        // s := Amount of ETH to stake at this time, s <= bufferEth + assets.
        //
        // Thus, the formula can be simplified to:
        // s = (totalQueueEth + bufferEth + assets) - (totalQueueEth + bufferEth
↪  + stakedEth + assets) * desiredBufferRatio
        //   = (totalQueueEth + bufferEth + assets) - targetBufferEth
        //
        // Flow:
        // If `s` <= 0, don't stake any ETH.
        // If `s` < bufferEth + assets, stake `s` amount of ETH.
        // If `s` >= bufferEth + assets, all available ETH can be staked in
↪  theory.
        // However, we cap the stake amount. This is to prevent the buffer from
↪  being completely drained.
        //
        // Let `a` be the available amount of ETH in the buffer after the
↪  deposit. `a` is calculated as:
        // a = (bufferEth + assets) - s
```


SHERLOCK

```
        uint256 targetBufferEth = ((totalAssets() + assets) *
↪  $.targetBufferPercentage) / BUFFER_PERCENTAGE_PRECISION;

        /// WRITE ///
        _mint(msg.sender, shares);

        uint256 availableEth = bufferEthCache + assets; // non-zero

        // If the buffer is insufficient or staking is paused, doesn't stake any
↪  of the deposit
        StakeLimitTypes.Data memory data =
↪  $.packedStakeLimitData.getStorageStakeLimitStruct();
        if (targetBufferEth >= availableEth + queueEthCache ||
↪  data.isStakingPaused()) {
            /// WRITE ///
            $.bufferEth = availableEth.toUint128();
            return (assets, shares);
        }

        // Calculate the amount of ETH to stake
        uint256 stakeAmount; // can be 0
        unchecked {
@>          stakeAmount = availableEth + queueEthCache - targetBufferEth; //
↪  non-zero, no underflow
        }
        // If the calculated stake amount exceeds the available ETH, simply
↪  assign the available ETH to the stake amount.
        // Possible scenarios:
        // - Target buffer percentage was changed to a lower value and there is
↪  a large withdrawal request pending.
        // - There is a pending withdrawal request and the available ETH are not
↪  left in the buffer.
        // - There is no pending withdrawal request and the available ETH are
↪  not left in the buffer.
        if (stakeAmount > availableEth) {
            // Note: Admins should be aware of this situation and take action to
↪  refill the buffer.
            // - Pause staking to prevent further staking until the buffer is
↪  refilled
            // - Update stake limit to a lower value
            // - Increase the target buffer percentage
@>          stakeAmount = availableEth; // All available ETH
        }

        // If the amount of ETH to stake exceeds the current stake limit, cap
↪  the stake amount.
```


SHERLOCK

```
        // This is to prevent the buffer from being completely drained. This is
↪   not a complete solution.
        uint256 currentStakeLimit =
↪   StakeLimitUtils.calculateCurrentStakeLimit(data); // can be 0 if the stake
↪   limit is exhausted
        if (stakeAmount > currentStakeLimit) {
@>          stakeAmount = currentStakeLimit;
        }
        /// WRITE ///
        // Update the stake limit state in the storage
        $.packedStakeLimitData.setStorageStakeLimitStruct(data.updatePrevStakeLi
↪   mit(currentStakeLimit - stakeAmount));

        /// INTERACT ///
        // Deposit into the yield source
        // Actual amount of ETH spent may be less than the requested amount.
@>      stakeAmount = _stake(stakeAmount); // stake amount can be 0

        /// WRITE ///
        $.bufferEth = (availableEth - stakeAmount).toUint128(); // no underflow
↪   theoretically

        return (assets, shares);
    }
```

The stakeAmount could be any small value. The users deposit right value using Tranche, but could revert, and they don't konw why.

Impact

The users deposit right value using Tranche, but could revert, and they don't konw why.

Code Snippet

https://github.com/sherlock-audit/2024-05-napier-update/blob/main/napier-uups-adapters/src/adapters/kelp/RsETHAdapter.sol#L77-L77

Tool used

Manual Review

Recommendation

```
 function _stake(uint256 stakeAmount) internal override returns (uint256) {
        if (stakeAmount == 0) return 0;

        // Check LRTDepositPool stake limit
        uint256 stakeLimit =
 ↪  RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH);
        if (stakeAmount > stakeLimit) {
            // Cap stake amount
            stakeAmount = stakeLimit;
        }
        // Check LRTDepositPool minAmountToDeposit
-        if (stakeAmount <= RSETH_DEPOSIT_POOL.minAmountToDeposit()) revert
 ↪  MinAmountToDepositError();
+        if (stakeAmount <= RSETH_DEPOSIT_POOL.minAmountToDeposit()) return 0;
        // Check paused of LRTDepositPool
        if (RSETH_DEPOSIT_POOL.paused()) revert ProtocolPaused();

        // Interact
        IWETH9(Constants.WETH).withdraw(stakeAmount);
        uint256 _rsETHAmt = RSETH.balanceOf(address(this));
        RSETH_DEPOSIT_POOL.depositETH{value: stakeAmount}(0, REFERRAL_ID);
        _rsETHAmt = RSETH.balanceOf(address(this)) - _rsETHAmt;

        if (_rsETHAmt == 0) revert InvariantViolation();

        return stakeAmount;
    }
```

Discussion

massun-onibakuchi

Such DoS doesn't meat requirements. This is because

- The revert doesn't always happen. It can happens when depositing small amount.
- It's unlikely that users deposit such small amount, wasting gas cost.
- `stakeAmount` can be changed by changing maxStakeLimit on adapter and we can definitely avoid such revert if needed

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

z3s commented:

> Low/Info; For an issue to be a valid Denial of Service (DoS), it must meet one of these criteria: 1. The issue causes locking of funds for users for more than a week. 2. The issue impacts the availability of time-sensitive functions. but The stakeAmount can be modified by changing the maxStakeLimit.

0502lian

it's not because It can happens when depositing small amount. It's because stakeAmount is calculated by `prefundedDeposit` . User deposit a large amount, stakeAmount can still be small amount (even zero) in _stake()

WangSecurity

After the discussions on escalation on #54, this report will be the main issue of a new family.

Issue M-8: Adapters revert when 0 shares are minted, making it impossible to deposit under certain conditions

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/64

Found by

no, zzykxx

Summary

Users are unable to deposit into an Adapter in some situations due to the `_stake()` function reverting.

Vulnerability Detail

The function `_stake()` in all of the in-scope adapters reverts if the amounts of minted shares of the targeted protocol is `0`.

Funds are deposited in an adapter via the prefundedDeposit() function, which internally calls `_stake()` by passing the amount to stake in the protocol, `stakeAmount`:

```
    ...SNIP...
    uint256 stakeAmount;
    unchecked {
        stakeAmount = availableEth + queueEthCache - targetBufferEth;
    }

    if (stakeAmount > availableEth) {
@>      stakeAmount = availableEth;
    }

    ...SNIP...
@>   stakeAmount = _stake(stakeAmount); // stake amount can be 0
    ...SNIP...
```

The amount to stake in the protocol, `stakeAmount`, can be restricted to `availableEth`. If `availableEth`/`stakeAmount` is low enough (but not `0`) for the targeted protocol to mint `0` shares all of the adapters in-scope will revert by throwing an `InvariantViolation();` error:

- UniEthAdapter
- EETHAdapter

- RsETHAdapter
- PufETHAdapter
- RenzoAdapter
- RswETHAdapter

Impact

Users won't not be able to deposit funds if the `stakeAmount` is not enough to mint at least 1 share. The protocol genrally allows users to deposit both when `stakeAmount` is `0` and when the maximum deposit cap has been reached on the target protocol, which is incosistent with the behaviour outlined in this report.

A similar finding was disclosed in the previous Napier contest.

Code Snippet

Tool used

Manual Review

Recommendation

The function `_stake()` in the adapters should ensure that the shares minted are at least `1` before actually depositing the funds. This might introduce a lot of overhead for the calculations, an alternative solution is to have the `_stake()` functions always return `0` if `stakeAmount` is lower than a certain (small) threshold:

```
function _stake(uint256 stakeAmount) internal override returns (uint256) {
    if (stakeAmount < 1e6) return 0;
    ...SNIP...
}
```

If going for a different fix please note that the EETHAdapter will actually revert on the internal call to `deposit()` if `0` shares are minted, instead of in the adapter.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/napier-uups-adapters/pull/16
https://github.com/napierfi/napier-v1/pull/220

zzykxx

An edge case was found in the proposed fix. `_stake()` could still revert if `RSETH_DEPOSIT_POOL.minAmountToDeposit()` returns `0`. This has been fixed in a new PR: https://github.com/napierfi/napier-uups-adapters/pull/23

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-9: Kelp adapter won't allow users to deposit if `getAssetCurrentLimit` returns `0`

Source: https://github.com/sherlock-audit/2024-05-napier-update-judging/issues/75

Found by

merlin, zzykxx

Summary

Users will be unable to deposit into the Kelp adapter when

Vulnerability Detail

The function `_stake()` in the RsETHAdapter::_stake() checks the current limits on `ETH` deposits of the Kelp protocol before depositing:

```
function _stake(uint256 stakeAmount) internal override returns (uint256) {
    if (stakeAmount == 0) return 0;

    uint256 stakeLimit = RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH);
    if (stakeAmount > stakeLimit) {
        stakeAmount = stakeLimit;
    }
    ...SNIP..
}
```

The returned value of the call to `RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH)` is assigned to the variable `stakeAmount`. This is the implementation of the called function, wich can be found here:

```
function getAssetCurrentLimit(address asset) public view override returns
↪  (uint256) {
    uint256 totalAssetDeposits = getTotalAssetDeposits(asset);
    if (totalAssetDeposits > lrtConfig.depositLimitByAsset(asset)) {
        return 0;
    }

    return lrtConfig.depositLimitByAsset(asset) - totalAssetDeposits;
}
```

As it can be seen it's possible for the function to return `0`, which will make the subsequent calls in the `_stake()` function revert. This is inconsistent with the general behaviour of the software, which always returns `0` when `stakeAmount` is `0`.

Impact

Users won't not be able to deposit funds in the `RsETHAdapter` because RsETHAdapter::_stake() will revert.

Code Snippet

Tool used

Manual Review

Recommendation

In RsETHAdapter::_stake() move the `if (stakeAmount == 0) return 0;` line after the assets limits have been queried:

```
function _stake(uint256 stakeAmount) internal override returns (uint256) {
    // Check LRTDepositPool stake limit
    uint256 stakeLimit = RSETH_DEPOSIT_POOL.getAssetCurrentLimit(Constants.ETH);
    if (stakeAmount > stakeLimit) {
        // Cap stake amount
        stakeAmount = stakeLimit;
    }

@>  if (stakeAmount == 0) return 0;

    // Check LRTDepositPool minAmountToDeposit
    if (stakeAmount <= RSETH_DEPOSIT_POOL.minAmountToDeposit()) return 0;
    // Check paused of LRTDepositPool
    if (RSETH_DEPOSIT_POOL.paused()) revert ProtocolPaused();

    // Interact
    IWETH9(Constants.WETH).withdraw(stakeAmount);
    uint256 _rsETHAmt = RSETH.balanceOf(address(this));
    RSETH_DEPOSIT_POOL.depositETH{value: stakeAmount}(0, REFERRAL_ID);
    _rsETHAmt = RSETH.balanceOf(address(this)) - _rsETHAmt;

    if (_rsETHAmt == 0) revert InvariantViolation();

    return stakeAmount;
}
```


SHERLOCK

RsETHAdapter.sol#L77

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/napierfi/napier-uups-adapters/pull/13

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.