SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Napier
Prepared by: Sherlock
Lead Security Expert: xiaoming90
Dates Audited: February 14 - February 26, 2024
Prepared on: April 22, 2024

Introduction

The liquidity hub for yield trading.

Scope

Repository: napierfi/v1-pool

Branch: main

Commit: 96689573e363667d920d59aa890b7b1f7418f4e8

Repository: napierfi/napier-v1

Branch: main

Commit: 31892e6ffecff018f2da25a45705857e509e0e11

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
13	4

Issues not fixed or acknowledged

Medium	High
0	0

Issue H-1: All yield could be drained if users set any `> 0` allowance to others

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/28

Found by

DenTonylifer, KingNFT, cawfree

Summary

`Tranche.redeemWithYT()` is not well implemented, all yield could be drained if users set any `> 0` allowance to others.

Vulnerability Detail

The issue arises on L283, all `accruedInTarget` is sent out, this will not work while users have allowances to others. Let's say, alice has `1000 YT` (yield token) which has generated `100 TT` (target token), and if she approves bob `100 YT` allowance, then bob should only be allowed to take the proportional target token, which is `100 TT * (100 YT / 1000 YT) = 10 TT`.

```
File: src\Tranche.sol
246:     function redeemWithYT(address from, address to, uint256 pyAmount)
↪  external nonReentrant returns (uint256) {
...
261:         accruedInTarget += _computeAccruedInterestInTarget(
262:             _gscales.maxscale,
263:             _lscale,
...
268:             _yt.balanceOf(from)
269:         );
..
271:         uint256 sharesRedeemed = pyAmount.divWadDown(_gscales.maxscale);
...
283:         _target.safeTransfer(address(adapter), sharesRedeemed +
↪  accruedInTarget);
284:         (uint256 amountWithdrawn, ) = adapter.prefundedRedeem(to);
...
287:         return amountWithdrawn;
288:     }
```

The following coded PoC shows all unclaimed and unaccrued target token could be drained out, even if the allowance is as low as `1wei`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import {TestTranche} from "./Tranche.t.sol";
import "forge-std/console2.sol";

contract TrancheAllowanceIssue is TestTranche {
    address bob = address(0x22);
    function setUp() public virtual override {
        super.setUp();
    }

    function testTrancheAllowanceIssue() public {
        // 1. issue some PT and YT
        deal(address(underlying), address(this), 1_000e6, true);
        tranche.issue(address(this), 1_000e6);

        // 2. generating some unclaimed yield
        vm.warp(block.timestamp + 30 days);
        _simulateScaleIncrease();

        // 3. give bob any negligible allowance, could be as low as only 1wei
        tranche.approve(bob, 1);
        yt.approve(bob, 1);

        // 4. all unclaimed and pending yield drained by bob
        assertEq(0, underlying.balanceOf(bob));
        vm.prank(bob);
        tranche.redeemWithYT(address(this), bob, 1);
        assertTrue(underlying.balanceOf(bob) > 494e6);
    }
}
```

And the logs:

```
2024-01-napier\napier-v1> forge test --match-test testTrancheAllowanceIssue -vv
[] Compiling...
[] Compiling 42 files with 0.8.19
[] Solc 0.8.19 finished in 82.11sCompiler run successful!
[] Solc 0.8.19 finished in 82.11s

Running 1 test for test/unit/TrancheAllowanceIssue.t.sol:TrancheAllowanceIssue
[PASS] testTrancheAllowanceIssue() (gas: 497585)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 11.06ms
```

```
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Impact

Users lost all unclaimed and unaccrued yield

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L283

Tool used

Manual Review

Recommendation

```
diff --git a/napier-v1/src/Tranche.sol b/napier-v1/src/Tranche.sol
index 62d9562..65db5c6 100644
--- a/napier-v1/src/Tranche.sol
+++ b/napier-v1/src/Tranche.sol
@@ -275,12 +275,15 @@ contract Tranche is BaseToken, ReentrancyGuard, Pausable,
 ↪  ITranche {
         delete unclaimedYields[from];
         gscales = _gscales;

+        uint256 accruedProportional = accruedInTarget * pyAmount /
 ↪  _yt.balanceOf(from);
+        unclaimedYields[from] = accruedInTarget - accruedProportional;
+
         // Burn PT and YT tokens from `from`
         _burnFrom(from, pyAmount);
         _yt.burnFrom(from, msg.sender, pyAmount);

         // Withdraw underlying tokens from the adapter and transfer them to the
 ↪  user
-        _target.safeTransfer(address(adapter), sharesRedeemed +
 ↪  accruedInTarget);
+        _target.safeTransfer(address(adapter), sharesRedeemed +
 ↪  accruedProportional);
         (uint256 amountWithdrawn, ) = adapter.prefundedRedeem(to);

         emit RedeemWithYT(from, to, amountWithdrawn);
```

Discussion

sherlock-admin

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

takarez commented:

> valid: high(1)

sherlock-admin3

The protocol team fixed this issue in PR/commit https://github.com/napierfi/napier-v1/pull/171.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-2: YT holder are unable to claim their interest

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/80

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

A malicious user could prevent YT holders from claiming their interest, leading to a loss of assets.

Vulnerability Detail

The interest accrued (in target token) by a user is computed based on the following formula, where $lscale$ is the user's last scale update and $maxScale$ is the max scale observed so far. The formula is taken from here:

$$interestAccrued = y(\frac{1}{lscale} - \frac{1}{maxScale})$$

A malicious user could perform the following steps to manipulate the $maxScale$ to a large value to prevent YT holders from claiming their interest.

1) When the Tranche and Adaptor are deployed, immediately mint the smallest possible number of shares. Attackers can do so by calling the Adaptor's `prefundedDeposit` function directly if they want to avoid issuance fees OR call the `Tranche.issue` function
2) Transfer large amounts of assets to the Adaptor directly. With a small amount of total supply and a large amount of total assets, the Adaptor's scale will be extremely large. Let the large scale at this point be L.
3) Trigger any function that will trigger an update to the global scale. The max scale will be updated to L and locked at this large scale throughout the entire lifecycle of the Tranche, as it is not possible for the scale to exceed L based on the current market yield or condition.
4) Attacker withdraws all their shares and assets via Adaptor's `prefundedDeposit` function or Tranche's redeem functions. Note that the attacker will not incur any fees during withdrawal. Thus, this attack is economically cheap and easy to execute.

5) After the attacker's withdrawal, the current scale of the adapter will revert back to normal (e.g. 1.0 exchange rate), and the scale will only increase due to the yield from staked ETH (e.g. daily rebase) and increase progressively (e.g., 1.0 > 1.05 > 1.10 > 1.15)

When a user issues/mints new PT and YT, their `lscales[user]` will be set to the $maxScale$, which is L.

A user's `lscales[user]` will only be updated if the adaptor's current scale is larger than L. As stated earlier, it is not possible for the scale to exceed L based on the current market yield or condition. Thus, the user's `lscales[user]` will be stuck at L throughout the entire lifecycle of the Tranche.

As a result, there is no way for the YT holder to claim their interest because, since $lscale == maxScale$, the interest accrued computed from the formula will always be zero. Thus, even if the Tranche/Adaptor started gaining yield from staked ETH, there is no way for YT holders to claim that.

Impact

YT holders cannot claim their interest, leading to a loss of assets.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L704

Tool used

Manual Review

Recommendation

Ensure that the YT holders can continue to claim their accrued interest earned within the Tranche/Adaptor regardless of the max scale.

Discussion

nevillehuang

Escalate

Duplicate of #92, exact same root cause, involving manipulations of maxScales

sherlock-admin2

> Escalate
>
> Duplicate of #92, exact same root cause, involving manipulations of maxScales

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

massun-onibakuchi

Escalate @nevillehuang
This issue should be lower. I think this is duplicate of #125. As you say, the root cause is that scale can be manipulated, which is mainly based on a kind of inflation attack. This attack assume an attacker is able to be the first depositor to manipulate rate effectively. Also at that time, basically no one has not issued any PT/YT. Even if someone mints some PT/YT, they can redeem them by burning together (`redeemWithYt`) with minimum issuance fee loss. This issue can be easily prevented by minting some shares earlier than a attacker in the same way as well-known inflation attack.

nevillehuang

@massun-onibakuchi In that case, I think medium severity is appropriate for this issue and #92, which should be duplicated together.

xiaoming9090

The sponsor mentioned that the mitigation would be to mint some shares in advance to prevent the well-known inflation attack. I agree with the sponsor that this is the correct approach to fix the issue.

However, the mitigation of minting some shares in advance to prevent this issue was not documented in the contest's README. Thus, the issue remains valid and as it is, as mitigation cannot be applied retrospectively after the contest. Otherwise, it would be unfair to the Watsons to raise an issue that only gets invalid by a mitigation shared after the contest/audit.

xiaoming9090

> Escalate @nevillehuang This issue should be lower. I think this is duplicate of #125. As you say, the root cause is that scale can be manipulated, which is mainly based on a kind of inflation attack. This attack assume an attacker is able to be the first depositor to manipulate rate effectively. Also at that time, basically no one has not issued any PT/YT. Even if someone mints some PT/YT, they can redeem them by burning together (`redeemWithYt`) with minimum issuance fee loss. This

> issue can be easily prevented by minting some shares earlier than a attacker in the same way as well-known inflation attack.

Disagree that this issue should be duplicated with Issue https://github.com/sherlock-audit/2024-01-napier-judging/issues/125.

Issue https://github.com/sherlock-audit/2024-01-napier-judging/issues/125 (and its duplicate https://github.com/sherlock-audit/2024-01-napier-judging/issues/94, which has a better write-up of the issue) discusses the classic vault inflation attack. This attack exploits the rounding error and exchange rate manipulation to steal from the victim depositor. It is a well-known attack that everyone is aware of.

This issue concerns manipulating the max scale by malicious users, which has prevented the YT holders from claiming their interest. This is a more complex issue and only specific to the Naiper protocol.

Thus, these two are different issues, and we should be careful not to overgeneralize all issues under the same category (e.g., re-entrancy, manipulation, access control) as one single issue without considering the details.

xiaoming9090

Escalate

This issue should be a High risk instead of Medium as this issue prevents YT holders from claiming their interest, leading to a loss of assets.

sherlock-admin2

> Escalate
>
> This issue should be a High risk instead of Medium as this issue prevents YT holders from claiming their interest, leading to a loss of assets.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cvetanovv

First, I disagree with the sponsor that this should be Low severity and a duplicate of the #125.

As I wrote in other reports, this "inflation attack" is not documented as a known issue.

I agree with the escalations of @xiaoming9090 and @nevillehuang. This issue has the same root cause as #92 and can be duplicated as a valid High.

Czar102

After some considerations, I think duplicating this issue with #92 makes sense by claiming that both of these are sourced in the ability to manipulate the scale by donations.

Please note that this doesn't mean that all "inflation attacks" are considered of the same root cause.

Czar102

Result: High Duplicate of #92

Accepting only the first escalation since it was the first accurate one.

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: accepted
- xiaoming9090: rejected

Issue H-3: LP Tokens always valued at 3 PTs

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/90

Found by

xiaoming90

Summary

LP Tokens are always valued at 3 PTs. As a result, users of the AMM pool might receive fewer assets/PTs than expected. The AMM pool might be unfairly arbitraged, resulting in a loss for the pool's LPs.

Vulnerability Detail

The Napier AMM pool facilitates trade between underlying assets and PTs. The PTs in the pool are represented by the Curve's Base LP Token of the Curve's pool that holds the PTs. The Napier AMM pool and Router math assumes that the Base LP Token is equivalent to 3 times the amount of PTs, as shown below. When the pool is initially deployed, it is correct that the LP token is equivalent to 3 times the amount of PT.

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/libs/PoolMath.sol#L106

```
File: PoolMath.sol
83:     int256 internal constant N_COINS = 3;
..SNIP..
96:     function swapExactBaseLpTokenForUnderlying(PoolState memory pool,
↪  uint256 exactBaseLptIn)
97:         internal
..SNIP..
105:             // Note: Here we are multiplying by N_COINS because the swap
↪  formula is defined in terms of the amount of PT being swapped.
106:             // BaseLpt is equivalent to 3 times the amount of PT due to the
↪  initial deposit of 1:1:1:1=pt1:pt2:pt3:Lp share in Curve pool.
107:             exactBaseLptIn.neg() * N_COINS
..SNIP..
120:     function swapUnderlyingForExactBaseLpToken(PoolState memory pool,
↪  uint256 exactBaseLptOut)
..SNIP..
125:         (int256 _netUnderlyingToAccount18, int256 _netUnderlyingFee18,
↪  int256 _netUnderlyingToProtocol18) = executeSwap(
126:             pool,
127:             // Note: sign is defined from the perspective of the swapper.
```

```
128:             // positive because the swapper is buying pt
129:             exactBaseLptOut.toInt256() * N_COINS
```

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierPool.sol#L48

```
File: NapierPool.sol
47:     /// @dev Number of coins in the BasePool
48:     uint256 internal constant N_COINS = 3;
..SNIP..
226:                     totalBaseLptTimesN: baseLptUsed * N_COINS,
..SNIP..
584:             totalBaseLptTimesN: totalBaseLpt * N_COINS,
```

In Curve, LP tokens are generally priced by computing the underlying tokens per share, hence dividing the total underlying token amounts by the total supply of the LP token. Given that the underlying assets in Curve's stable swap are pegged to each other, the invariant's D value can be computed to estimate the total value of the underlying tokens.

Curve itself provides a function `get_virtual_price` that computes the price of the LP token by dividing D with the total supply.

Note that for LP tokens, the ratio of the total underlying value and the total supply will grow (fee mechanism) over time. Thus, the virtual price's value will increase over time.

This means the LP token will be worth more than 3 PTs in the Curve Pool over time. However, the Naiper AMM pool still values its LP token at a constant value of 3 PTs. This discrepancy between the value of the LP tokens in the Napier AMM pool and Curve pool might result in various issues, such as the following:

- Investors brought LP tokens at the price of 3.X PT from the market. The LP tokens are deposited into or swap into the Napier AMM pool. The Naiper Pool will always assume that the price of the LP token is 3 PTs, thus shortchanging the number of assets or PTs returned to users.
- Potential arbitrage opportunity where malicious users obtain the LP token from the Naiper AMM pool at a value of 3 PT and redeem the LP token at a value higher than 3 PTs, pocketing the differences.

Impact

Users of the AMM pool might receive fewer assets/PTs than expected. The AMM pool might be unfairly arbitraged, resulting in a loss for the pool's LPs.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/libs/PoolMath.sol#L106

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierPool.sol#L48

Tool used

Manual Review

Recommendation

Naiper and Pendle share the same core math for their AMM pool.

In Pendle, the AMM stores the PTs and SY (Standard Yield Token). When performing any operation (e.g., deposit, swap), the SY will be converted to the underlying assets based on SY's current rate before performing any math operation. If it is a SY (wstETH), the SY's rate will be the current exchange rate for wstETH to stETH/ETH. One could also think the AMM's reserve is PTs and Underlying Assets under the hood.

In Napier, the AMM stores the PTs and Curve's LP tokens. When performing any operation, the math will always convert the LP token to underlying assets using a static exchange rate of 3. However, this is incorrect, as the value of an LP token will grow over time. The AMM should value the LP tokens based on their current value. The virtual price of the LP token and other information can be leveraged to derive the current value of the LP tokens to facilitate the math operation within the pool.

Discussion

sherlock-admin

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

takarez commented:

> valid; high(9)

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/napierfi/v1-pool/pull/159.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-4: Victim's fund can be stolen due to rounding error and exchange rate manipulation

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/94

The protocol has acknowledged this issue.

Found by

Bandit, LTDingZhen, cawfree, jennifer37, xAlismx, xiaoming90

Summary

Victim's funds can be stolen by malicious users by exploiting the rounding error and through exchange rate manipulation.

Vulnerability Detail

The LST Adaptor attempts to guard against the well-known vault inflation attack by reverting the TX when the amount of shares minted is rounded down to zero in Line 78 below.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L71

```
File: BaseLSTAdapter.sol
71:     function prefundedDeposit() external nonReentrant returns (uint256,
↪  uint256) {
72:         uint256 bufferEthCache = bufferEth; // cache storage reads
73:         uint256 queueEthCache = withdrawalQueueEth; // cache storage reads
74:         uint256 assets = IWETH9(WETH).balanceOf(address(this)) -
↪  bufferEthCache; // amount of WETH deposited at this time
75:         uint256 shares = previewDeposit(assets);
76:
77:         if (assets == 0) return (0, 0);
78:         if (shares == 0) revert ZeroShares();
```

However, this control alone is not sufficient to guard against vault inflation attacks.

Let's assume the following scenario (ignoring fee for simplicity's sake):

1. The victim initiates a transaction that deposits 10 ETH as the underlying asset when there are no issued estETH shares.
2. The attacker observes the victim's transaction and deposits 1 wei of ETH (issuing 1 wei of estETH share) before the victim's transaction. 1 wei of estETH

share worth of PT and TY will be minted to the attacker.

3. Then, the attacker executes a transaction to directly transfer 5 stETH to the adaptor. The exchange rate at this point is `1 wei / (5 ETH + 1 wei)`. Note that the totalAssets function uses the `balanceOf` function to compute the total underlying assets owned by the adaptor. Thus, this direct transfer will increase the total assets amount.
4. When the victim's transaction is executed, the number of estETH shares issued is calculated as `10 ETH * 1 wei / (5 ETH + 1 wei)`, resulting in 1 wei being issued due to round-down.
5. The attacker will combine the PT + YT obtained earlier to redeem 1 wei of estETH share from the adaptor.
6. The attacker, holding 50% of the issued estETH shares (indirectly via the PT+YT he owned), receives `(15 ETH + 1 wei) / 2` as the underlying asset.
7. The attacker seizes 25% of the underlying asset (2.5 ETH) deposited by the victim.

This scenario demonstrates that even when a revert is triggered due to the number of issued estETH share being 0, it does not prevent the attacker from capturing the user's funds through exchange rate manipulation.

Impact

Loss of assets for the victim.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L71

Tool used

Manual Review

Recommendation

Following are some of the measures that could help to prevent such an attack:

- Mint a certain amount of shares to zero address (dead address) during contract deployment (similar to what has been implemented in Uniswap V2)
- Avoid using the `balanceOf` so that malicious users cannot transfer directly to the contract to increase the assets per share. Track the total assets internally via a variable.

Discussion

massun-onibakuchi

Openzeppelin's ERC4626 with a decimalsOffset=0 in the virtual share mitigates the issue, but it is recognized that it does not completely resolve it. We plan to deposit a small amount of tokens after deployment.

ydspa

Escalate This finding is not high. This attack would be success only if the protocol team doesn't provide and keep any initial liquidity for the vault. However, in practical, we merely find protocol teams don't provide any fund to bootstrap projects. The likelihood of this attack is very low.

sherlock-admin2

> Escalate This finding is not high. This attack would be success only if the protocol team doesn't provide and keep any initial liquidity for the vault. However, in practical, we merely find protocol teams don't provide any fund to bootstrap projects. The likelihood of this attack is very low.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

massun-onibakuchi

> Escalate This finding is not high. This attack would be success only if the protocol team doesn't provide and keep any initial liquidity for the vault. However, in practical, we merely find protocol teams don't provide any fund to bootstrap projects. The likelihood of this attack is very low.

Agree. This kind of inflation attack is not listed as a finding in the Mixbytes audit, which is still unpublished.

MehdiKarimi81

If the protocol team doesn't provide initial liquidity it would be high, since it's not listed as a known issue and the protocol team didn't clear that they plan to provide initial liquidity, it can be considered as a high severity.

xiaoming9090

The mitigation of minting some shares in advance to prevent this issue was not documented in the contest's README. Thus, the issue remains valid and as it is, as mitigation cannot be applied retrospectively after the contest. Otherwise, it would be unfair to the Watsons to raise an issue that only gets invalid by a mitigation shared after the contest/audit.

cvetanovv

I disagree with the escalation and this report should remain a valid High. Nowhere in the Readme or Documentation is it described that the protocol knows and can prevent this attack.

Czar102

Uncommunicated plans for issue mitigation don't constitute a reason for invalidation. I believe this was correctly judged – planning to reject the escalation and leave the issue as is.

Czar102

Result: High Has duplicates

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- ydspa: rejected

Issue M-1: Napier pool owner can unfairly increase protocol fees on swaps to earn more revenue

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/51

The protocol has acknowledged this issue.

Found by

Solidity_ATL_Team_2, xiaoming90

Summary

Currently there is no limit to how often a `poolOwner` can update fees which can be abused to earn more fees by charging users higher swap fees than they expect.

Vulnerability Detail

The `NapierPool::setFeeParameter` function allows the `poolOwner` to set the `protocolFeePercent` at any point to a maximum value of 100%. The `poolOwner` is a trusted party but should not be able to abuse protocol settings to earn more revenue. There are no limits to how often this can be updated.

Impact

A malicious `poolOwner` could change the protocol swap fees unfairly for users by front-running swaps and increasing fees to higher values on unsuspecting users. An example scenario is:

- The `poolOwner` sets swap fees to 1% to attract users
- The `poolOwner` front runs all swaps and changes the swap fees to the maximum value of 100%
- After the swap the `poolOwner` resets `protocolFeePercent` to a low value to attract more users

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierPool.sol#L544-L556 https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/libs/PoolMath.sol#L313

Tool used

Manual Review and Foundry

Proof of concept

```
function test_protocol_owner_frontRuns_swaps_with_higher_fees() public
↪  whenMaturityNotPassed {
       // pre-condition
       vm.warp(maturity - 30 days);
       deal(address(pts[0]), alice, type(uint96).max, false); // ensure alice
↪  has enough pt
       uint256 preBaseLptSupply = tricrypto.totalSupply();
       uint256 ptInDesired = 100 * ONE_UNDERLYING;
       uint256 expectedBaseLptIssued = tricrypto.calc_token_amount([ptInDesired,
↪  0, 0], true);

       // Pool owner sees swap about to occur and front runs updating fees to
↪  max value
       vm.startPrank(owner);
       pool.setFeeParameter("protocolFeePercent", 100);
       vm.stopPrank();

       // execute
       vm.prank(alice);
       uint256 underlyingOut = pool.swapPtForUnderlying(
           0, ptInDesired, recipient,
↪  abi.encode(CallbackInputType.SwapPtForUnderlying, SwapInput(underlying,
↪  pts[0]))
       );
       // sanity check
       uint256 protocolFee = SwapEventsLib.getProtocolFeeFromLastSwapEvent(pool);
       assertGt(protocolFee, 0, "fee should be charged");
   }
```

Recommendation

Introduce a delay in fee updates to ensure users receive the fees they expect.

Discussion

sherlock-admin

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

takarez commented:

invalid

massun-onibakuchi

we have acknowledge the issue

cvetanovv

As the sponsor wrote in the other report, slippage protection can prevent a malicious increase in fees.

Darkartt

Escalate

sherlock-admin2

Escalate

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Robert-H-Leonard

There is slippage protection in the swap but it does not fully protect against a malicious increase.

The PoC in this issue can be added to the test suite in `v1-pool/test/unit/pool/Swap.t.sol` to show this increase is possible. The default protocol fee is set to 80% and this test is setting it to 100% and performing the swap.

To test an extreme case that still passes you can decrease the default protocol fee to 5% (which is configured here and still have the fee increase.

Czar102

@cvetanovv can you elaborate on your stance, also given the new context?

cvetanovv

I agree with the escalation. This report and #98 should be Medium.

Czar102

Planning to apply the suggestion. Any idea why is the escalation empty? I'm not sure if I should accept it or choose this one to be accepted for this single modification of the validity.

Robert-H-Leonard

@Czar102 the escalation is empty because we are working on a team of 5 and he was the only team member that could raise it. My comment below his escalation is the context of the escalation

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- Darkartt: accepted

Czar102

Result: Medium Has duplicates

Czar102

@Robert-H-Leonard next time please just share the escalation contents with each other to put the recommendation in the escalation. This won't be accepted in the future.

Issue M-2: Benign esfrxETH holders incur more loss than expected

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/82

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

Malicious esfrxETH holders can avoid "pro-rated" loss and have the remaining esfrxETH holders incur all the loss due to the fee charged by FRAX during unstaking. As a result, the rest of the esfrxETH holders incur more losses than expected compared to if malicious esfrxETH holders had not used this trick in the first place.

Vulnerability Detail

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/frax/SFrxETHAdapter.sol#L22

```
File: SFrxETHAdapter.sol
17: /// @title SFrxETHAdapter - esfrxETH
18: /// @dev Important security note:
19: /// 1. The vault share price (esfrxETH / WETH) increases as sfrxETH accrues
 ↪  staking rewards.
20: /// However, the share price decreases when frxETH (sfrxETH) is withdrawn.
21: /// Withdrawals are processed by the FraxEther redemption queue contract.
22: /// Frax takes a fee at the time of withdrawal requests, which temporarily
 ↪  reduces the share price.
23: /// This loss is pro-rated among all esfrxETH holders.
24: /// As a mitigation measure, we allow only authorized rebalancers to request
 ↪  withdrawals.
25: ///
26: /// 2. This contract doesn't independently keep track of the sfrxETH
 ↪  balance, so it is possible
27: /// for an attacker to directly transfer sfrxETH to this contract, increase
 ↪  the share price.
28: contract SFrxETHAdapter is BaseLSTAdapter, IERC721Receiver {
```

In the `SFrxETHAdapter`'s comments above, it is stated that the share price will decrease due to the fee taken by FRAX during the withdrawal request. This loss is

supposed to be 'pro-rated' among all esfrxETH holders. However, this report reveals that malicious esfrxETH holders can circumvent this 'pro-rated' loss, leaving the remaining esfrxETH holders to bear the entire loss. Furthermore, the report demonstrates that the current mitigation measure, which allows only authorized rebalancers to request withdrawals, is insufficient to prevent this exploitation.

Whenever a rebalancers submit a withdrawal request to withdraw staked ETH from FRAX, it will first reside in the mempool of the blockchain and anyone can see it. Malicious esfrxETH holders can front-run it to withdraw their shares from the adaptor.

When the withdrawal request TX is executed, the remaining esfrxETH holders in the adaptor will incur the fee. Once executed, the malicious esfrxETH deposits back to the adaptors.

Note that no fee is charged to the users for any deposit or withdrawal operation. Thus, as long as the gain from this action is more than the gas cost, it makes sense for the esfrxETH holders to do so.

Impact

The rest of the esfrxETH holders incur more losses than expected compared to if malicious esfrxETH holders had not used this trick in the first place.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/frax/SFrxETHAdapter.sol#L22

Tool used

Manual Review

Recommendation

The best way to discourage users from withdrawing their assets and depositing them back to take advantage of a particular event is to impose a fee upon depositing and withdrawing.

Discussion

massun-onibakuchi

We'll submit transactions using private rpc like flashbot.

xiaoming9090

Escalate.

Agree with the sponsor that this attack would be prevented if the withdraw request transaction is submitted via private RPC like flash-bot.

However, this attack and its mitigation (using a private RPC) were not highlighted as a known issue under the contest's README.

Thus, this issue should be considered valid and not excluded, as the mitigation measures are shared post-audit.

sherlock-admin2

> Escalate.
>
> Agree with the sponsor that this attack would be prevented if the withdraw request transaction is submitted via private RPC like flash-bot.
>
> However, this attack and its mitigation (using a private RPC) were not highlighted as a known issue under the contest's README.
>
> Thus, this issue should be considered valid and not excluded, as the mitigation measures are shared post-audit.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cvetanovv

I think @xiaoming9090 is right here. Nowhere is it described that private rpc like flashbot will be used. So in my opinion it can be valid Medium.

Czar102

Planning to accept the escalation and make the issue a valid Medium.

Czar102

Result: Medium Unique

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- xiaoming9090: accepted

Issue M-3: Anyone can convert someone's unclaimed yield to PT + YT

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/83

Found by

KingNFT, xiaoming90

Summary

Anyone can convert someone's unclaimed yield to PT + YT, leading to a loss of assets for the victim.

Vulnerability Detail

Assume that Alice has accumulated 100 Target Tokens in her account's unclaimed yields. She is only interested in holding the Target token (e.g., she is might be long Target token). She intends to collect those Target Tokens sometime later.

Bob could disrupt Alice's plan by calling `issue` function with the parameter (`to=Alice, underlyingAmount=0`). The function will convert all 100 Target Tokens stored within Alice's account's unclaimed yield to PT + YT and send them to her, which Alice does not want or need in the first place.

Line 196 below will clear Alice's entire unclaimed yield before computing the accrued interest at Line 203. The accrued interest will be used to mint the PT and YT (Refer to Line 217 and Line 224 below).

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L179

```
File: Tranche.sol
179:     function issue(
180:         address to,
181:         uint256 underlyingAmount
182:     ) external nonReentrant whenNotPaused notExpired returns (uint256
↪  issued) {
..SNIP..
196:         lscales[to] = _maxscale;
197:         delete unclaimedYields[to];
198:
199:         uint256 yBal = _yt.balanceOf(to);
200:         // If recipient has unclaimed interest, claim it and then reinvest
↪  it to issue more PT and YT.
```

```
201:         // Reminder: lscale is the last scale when the YT balance of the
↪  user was updated.
202:         if (_lscale != 0) {
203:             accruedInTarget += _computeAccruedInterestInTarget(_maxscale,
↪  _lscale, yBal);
204:         }
..SNIP..
217:         uint256 sharesUsed = sharesMinted + accruedInTarget;
218:         uint256 fee = sharesUsed.mulDivUp(issuanceFeeBps, MAX_BPS);
219:         issued = (sharesUsed - fee).mulWadDown(_maxscale);
220:
221:         // Accumulate issueance fee in units of target token
222:         issuanceFees += fee;
223:         // Mint PT and YT to user
224:         _mint(to, issued);
225:         _yt.mint(to, issued);
```

The market value of the PT + YT might be lower than the market value of the Target Token. In this case, Alice will lose due to Bob's malicious action.

Another issue is that when Bob calls `issue` function on behalf of Alice's account, the unclaimed target tokens will be subjected to the issuance fee (See Line 218 above). Thus, even if the market value of PT + YT is exactly the same as that of the Target Token, Alice is still guaranteed to suffer a loss from Bob's malicious action.

If Alice had collected the unclaimed yield via the collect function, she would have received the total value of the yield in the underlying asset terms, as a collection of yield is not subjected to any fee.

Impact

Loss of assets for the victim.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L179

Tool used

Manual Review

Recommendation

Consider not allowing anyone to issue PY+YT on behalf of someone's account.

Discussion

sherlock-admin

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

takarez commented:

> invalid: he saves her a fee by doing so; similar: https://discordapp.com/channels/812037309376495636/1192483776622759956/1199612656395505684

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/napierfi/napier-v1/pull/178.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-4: Lack of slippage control for `issue` function

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/84

Found by

xiaoming90

Summary

The lack of slippage control for `issue` function can lead to a loss of assets for the affected users.

Vulnerability Detail

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 at Line 208 below. 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.

Having said that, the number of shared minted will vary (larger or smaller than expected) if there is a change in the current scale. Assuming that Alice determined off-chain that depositing 100 ETH would issue x amount of PT/YT. When she executes the TX, the scale increases, leading to the amount of PT/YT issued being less than x. The slippage is more than what she can accept.

In summary, the `issue` function lacks the slippage control that allows the users to revert if the amount of PT/YT they received is less than the amount they expected.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L179

```
File: Tranche.sol
179:     function issue(
180:         address to,
181:         uint256 underlyingAmount
182:     ) external nonReentrant whenNotPaused notExpired returns (uint256
↪  issued) {
..SNIP..
```

```
206:         // Transfer underlying from user to adapter and deposit it into
 ↪  adapter to get target token
207:         _underlying.safeTransferFrom(msg.sender, address(adapter),
 ↪  underlyingAmount);
208:         (, uint256 sharesMinted) = adapter.prefundedDeposit();
209:
210:         // Deduct the issuance fee from the amount of target token minted +
 ↪  reinvested yield
211:         // Fee should be rounded up towards the protocol (against the user)
 ↪  so that issued principal is rounded down
212:         // Hackmd: F0
213:         // ptIssued
214:         // = (u/s + y - fee) * S
215:         // = (sharesUsed - fee) * S
216:         // where u = underlyingAmount, s = current scale, y = reinvested
 ↪  yield, S = maxscale
217:         uint256 sharesUsed = sharesMinted + accruedInTarget;
218:         uint256 fee = sharesUsed.mulDivUp(issuanceFeeBps, MAX_BPS);
219:         issued = (sharesUsed - fee).mulWadDown(_maxscale);
220:
221:         // Accumulate issueance fee in units of target token
222:         issuanceFees += fee;
223:         // Mint PT and YT to user
224:         _mint(to, issued);
225:         _yt.mint(to, issued);
226:
227:         emit Issue(msg.sender, to, issued, sharesUsed);
228:     }
229:
```

Impact

Loss of assets for the affected users.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L179

Tool used

Manual Review

Recommendation

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

Discussion

sherlock-admin

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

takarez commented:

> valid: slippage; medium(15)

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/napierfi/v1-pool/pull/157.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-5: Users unable to withdraw their funds due to FRAX admin action

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/95

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

FRAX admin action can lead to the fund of Naiper protocol and its users being stuck, resulting in users being unable to withdraw their assets.

Vulnerability Detail

Per the contest page, the admins of the protocols that Napier integrates with are considered "RESTRICTED". This means that any issue related to FRAX's admin action that could negatively affect Napier protocol/users will be considered valid in this audit contest.

> Q: Are the admins of the protocols your contracts integrate with (if any) TRUSTED or RESTRICTED? RESTRICTED

When the Adaptor needs to unstake its staked ETH to replenish its ETH buffer so that users can redeem/withdraw their funds, it will first join the FRAX's redemption queue, and the queue will issue a redemption NFT afterward. After a certain period, the adaptor can claim their ETH by burning the redemption NFT at Line 65 via the `burnRedemptionTicketNft` function.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/frax/SFrxETHAdapter.sol#L65

```
File: SFrxETHAdapter.sol
53:     function claimWithdrawal() external override {
54:         uint256 _requestId = requestId;
55:         uint256 _withdrawalQueueEth = withdrawalQueueEth;
56:         if (_requestId == 0) revert NoPendingWithdrawal();
57:
58:         /// WRITE ///
59:         delete withdrawalQueueEth;
60:         delete requestId;
61:         bufferEth += _withdrawalQueueEth.toUint128();
62:
```

```
63:         /// INTERACT ///
64:         uint256 balanceBefore = address(this).balance;
65:         REDEMPTION_QUEUE.burnRedemptionTicketNft(_requestId, payable(this));
66:         if (address(this).balance < balanceBefore + _withdrawalQueueEth)
 ↪  revert InvariantViolation();
67:
68:         IWETH9(Constants.WETH).deposit{value: _withdrawalQueueEth}();
69:     }
```

However, it is possible for FRAX's admin to disrupt the redemption process of the adaptor, resulting in Napier users being unable to withdraw their funds. When the `burnRedemptionTicketNft` function is executed, the redemption NFT will be burned, and native ETH residing in the `FraxEtherRedemptionQueue` contract will be sent to the adaptor at Line 498 below

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3761

```
File: FraxEtherRedemptionQueue.sol
473:     function burnRedemptionTicketNft(uint256 _nftId, address payable
 ↪  _recipient) external nonReentrant {
..SNIP..
494:         // Effects: Burn frxEth to match the amount of ether sent to user
 ↪  1:1
495:         FRX_ETH.burn(_redemptionQueueItem.amount);
496:
497:         // Interactions: Transfer ETH to recipient, minus the fee
498:         (bool _success, ) = _recipient.call{ value:
 ↪  _redemptionQueueItem.amount }("");
499:         if (!_success) revert InvalidEthTransfer();
```

FRAX admin could execute the `recoverEther` function to transfer out all the Native ETH residing in the `FraxEtherRedemptionQueue` contract, resulting in the NFT redemption failing due to lack of ETH.

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3381

```
File: FraxEtherRedemptionQueue.sol
185:     /// @notice Recover ETH from exits where people early exited their NFT
 ↪  for frxETH, or when someone mistakenly directly sends ETH here
186:     /// @param _amount Amount of ETH to recover
187:     function recoverEther(uint256 _amount) external {
188:         _requireSenderIsTimelock();
189:
190:         (bool _success, ) = address(msg.sender).call{ value: _amount }("");
```

```
191:         if (!_success) revert InvalidEthTransfer();
192:
193:         emit RecoverEther({ recipient: msg.sender, amount: _amount });
194:     }
```

As a result, Napier users will not be able to withdraw their funds.

Impact

The fund of Naiper protocol and its users will be stuck, resulting in users being unable to withdraw their assets.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/frax/SFrxETHAdapter.sol#L65

Tool used

Manual Review

Recommendation

Ensure that the protocol team and its users are aware of the risks of such an event and develop a contingency plan to manage it.

Discussion

sherlock-admin

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

takarez commented:

> valid: per contest ReadMe; this should be valid; medium(11)

Issue M-6: `withdraw` function does not comply with ERC5095

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/96

Found by

0xVolodya, xiaoming90

Summary

The `withdraw` function of Tranche/PT does not comply with ERC5095 as it does not return the exact amount of underlying assets requested by the users.

Vulnerability Detail

Per the contest's README page, it stated that the code is expected to comply with ERC5095 (https://eips.ethereum.org/EIPS/eip-5095). As such, any non-compliance to ERC5095 found during the contest is considered valid.

> Q: Is the code/contract expected to comply with any EIPs? Are there specific assumptions around adhering to those EIPs that Watsons should be aware of? EIP20 and IERC5095

Following is the specification of the `withdraw` function of ERC5095. It stated that the user must receive exactly `underlyingAmount` of underlying tokens.

> withdraw Burns principalAmount from holder and sends exactly underlyingAmount of underlying tokens to receiver.

However, the `withdraw` function does not comply with this requirement.

On a high-level, the reason is that Line 337 will compute the number of shares that need to be redeemed to receive `underlyingAmount` number of underlying tokens from the adaptor. The main problem here is that the division done here is rounded down. Thus, the `sharesRedeem` will be lower than expected. Consequently, when `sharesRedeem` number of shares are redeemed at Line 346 below, the users will not receive an exact number of `underlyingAmount` of underlying tokens.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L328

```
File: Tranche.sol
328:     function withdraw(
329:         uint256 underlyingAmount,
330:         address to,
331:         address from
332:     ) external override nonReentrant expired returns (uint256) {
```

```
333:         GlobalScales memory _gscales = gscales;
334:         uint256 cscale = _updateGlobalScalesCache(_gscales);
335:
336:         // Compute the shares to be redeemed
337:         uint256 sharesRedeem = underlyingAmount.divWadDown(cscale);
338:         uint256 principalAmount = _computePrincipalTokenRedeemed(_gscales,
↪  sharesRedeem);
339:
340:         // Update the global scales
341:         gscales = _gscales;
342:         // Burn PT tokens from `from`
343:         _burnFrom(from, principalAmount);
344:         // Withdraw underlying tokens from the adapter and transfer them to
↪  `to`
345:         _target.safeTransfer(address(adapter), sharesRedeem);
346:         (uint256 underlyingWithdrawn, ) = adapter.prefundedRedeem(to);
```

Impact

The tranche/PT does not align with the ERC5095 specification.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L328

Tool used

Manual Review

Recommendation

Update the `withdraw` function to send exactly `underlyingAmount` number of underlying tokens to the caller so that the Tranche will be aligned with the ERC5095 specification.

Discussion

sherlock-admin

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

takarez commented:

> valid: medium(4)

sherlock-admin3

The protocol team fixed this issue in PR/commit https://github.com/napierfi/napier-v1/pull/170.

massun-onibakuchi

@nevillehuang I think this finding severity can be much lower. Actually EIP5095 is stagnant not even final status, which means it could be changed. we have not known protocols which uses this standard.

https://eips.ethereum.org/

Banditx0x

Escalate

This is low/informational. Zero impact, does not break core functionality, or any functionality for that matter.

sherlock-admin2

> Escalate
>
> This is low/informational. Zero impact, does not break core functionality, or any functionality for that matter.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

@Banditx0x Seems to be right according to this sherlock rule:

> EIP Compliance: For issues related to EIP compliance, the protocol & codebase must show that there are important external integrations that would require strong compliance with the EIP's implemented in the code. The EIP must be in regular use or in the final state for EIP implementation issues to be considered valid

xiaoming9090

Per the contest's README page, the protocol **explicitly** states that the code is expected to comply with ERC5095 (https://eips.ethereum.org/EIPS/eip-5095). Thus, any non-compliance to ERC5095 found by the Watsons during the contest is considered a valid Medium.

Otherwise, it would be unfair to all the Watson who made an effort to verify if the contracts comply with ERC5095.

> Q: Is the code/contract expected to comply with any EIPs? Are there specific assumptions around adhering to those EIPs that Watsons should be aware of? EIP20 and IERC5095

Note that the Hierarchy of truth in Sherlock's contest, "Contest README", precedes everything else, including "Sherlock rules for valid issues"

> Hierarchy of truth: Contest README > Sherlock rules for valid issues > protocol documentation (including code comments) > protocol answers on the contest public Discord channel.

Banditx0x

There are plenty of informational/low issues that don't get rewarded. The mention of an EIP in the ReadMe does not automatically boost an issue to Medium.

cvetanovv

For me, this issue is borderline medium/low. In the past, such reports have been Medium, but I also agree with @Banditx0x comment that this is more like Low severity.

Czar102

I am inclined to leave the issue as is, but will consult team members before sharing a preliminary verdict.

Czar102

After discussing this, I'm planning to reject the escalation and leave the issue as is. It will be made more explicit in the rules that issues about mentioned EIPs are valid.

Czar102

Result: Medium Has duplicates

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- banditx0x: rejected

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-7: Users are unable to collect their yield if tranche is paused

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/97

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

Users are unable to collect their yield if Tranche is paused, resulting in a loss of assets for the victims.

Vulnerability Detail

Per the contest's README page, it stated that the admin/owner is "RESTRICTED". Thus, any finding showing that the owner/admin can steal a user's funds, cause loss of funds or harm to the users, or cause the user's fund to be struck is valid in this audit contest.

> Q: Is the admin/owner of the protocol/contracts TRUSTED or RESTRICTED?
>
> RESTRICTED

The admin of the protocol has the ability to pause the Tranche contract, and no one except for the admin can unpause it. If a malicious admin paused the Tranche contract, the users will not be able to collect their yield earned, leading to a loss of assets for them.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L605

```
File: Tranche.sol
603:     /// @notice Pause issue, collect and updateUnclaimedYield
604:     /// @dev only callable by management
605:     function pause() external onlyManagement {
606:         _pause();
607:     }
608:
609:     /// @notice Unpause issue, collect and updateUnclaimedYield
610:     /// @dev only callable by management
611:     function unpause() external onlyManagement {
```

```
612:         _unpause();
613:     }
```

The following shows that the `collect` function can only be executed when the system is not paused.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L399

```
File: Tranche.sol
399:     function collect() public nonReentrant whenNotPaused returns (uint256) {
400:         uint256 _lscale = lscales[msg.sender];
401:         uint256 accruedInTarget = unclaimedYields[msg.sender];
```

Impact

Users are unable to collect their yield if Tranche is paused, resulting in a loss of assets for the victims.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L605

Tool used

Manual Review

Recommendation

Consider allowing the users to collect yield even when the system is paused.

Discussion

nevillehuang

Escalate

As mentioned by the watson, any issue that can causes a possible DoS/loss of funds by protocols admin not arising from external contract pauses/emergency withdrawals should be a valid medium severity issues due to centralization risks. In this case, protocol admins can block collection of yield permanently. In fact, it also blocks reinvestment of yield via `issue()`

sherlock-admin2

> Escalate
>
> As mentioned by the watson, any issue that can causes a possible DoS/loss of funds by protocols admin not arising from external contract pauses/emergency withdrawals should be a valid medium severity issues due to centralization risks. In this case, protocol admins can block collection of yield permanently. In fact, it also blocks reinvestment of yield via `issue()`

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

Agree with the escalation by Nevi. The report highlighted a way for the admin to pause the contract, resulting in the users not being able to collect their yield earned, leading to a loss of assets for them. Per the contest rules, such an issue is considered valid.

> Q: Is the admin/owner of the protocol/contracts TRUSTED or RESTRICTED?
>
> RESTRICTED

cvetanovv

The reason I left it invalid is that I consider the pausing mechanism a design decision.

Also, they have written in the readme that it is acceptable to have contracts pausing. Yes, I know this applies to External integrations, but I think it may be valid for them as well.

Apart from that pausing only stops users from collecting their rewards, doesn't mean the protocol will steal them. We have a sentence in the rules that gives the judge in this situation some flexibility to decide: "Please note that these restrictions must be explicitly described by the protocol and will be considered case by case."

xiaoming9090

> The reason I left it invalid is that I consider the pausing mechanism a design decision.
>
> Also, they have written in the readme that it is acceptable to have contracts pausing. Yes, I know this applies to External integrations, but I think it may be valid for them as well.

> Apart from that pausing only stops users from collecting their rewards, doesn't mean the protocol will steal them. We have a sentence in the rules that gives the judge in this situation some flexibility to decide: "Please note that these restrictions must be explicitly described by the protocol and will be considered case by case."

I agree that having a pausing mechanism is a design choice by the protocol team. However, that does not mean the malicious admin will not use this pausing mechanism to block users from collecting their yields, causing harm to them.

The contest's README stated that pausing by external protocols is fine, but that does not mean that internal pausing is out-of-scope during the contest.

When users cannot collect their earned yield due to malicious admin activities, it is basically the same as a loss of assets for them. Loss of assets for either users or protocols is considered a valid issue here.

Note: Admin is restricted in this contest.

Czar102

I agree that this is a Medium severity issue, planning to accept the escalation. It is clear that when the protocol team considers the admins "RESTRICTED" and doesn't post the restrictions, the owners should not be able to cause losses to users.

@cvetanovv The pausing mechanism is a design decision, but when it starts to allow to cause loss of funds by an untrusted actor, then it becomes a vulnerability, too. They are fine with external integration pausing, meaning that they trust the protocols to do it with the good of users in mind. This has an odd relation with the fact that external admins are restricted, but luckily we are not considering external admins here.

Czar102

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: accepted

Issue M-8: Permissioned rebalancing functions leading to loss of assets

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/99

The protocol has acknowledged this issue.

Found by

Arabadzhiev, ZanyBonzy, cawfree, thisvishalsingh, xiaoming90

Summary

Permissioned rebalancing functions that could only be accessed by admin could lead to a loss of assets.

Vulnerability Detail

Per the contest's README page, it stated that the admin/owner is "RESTRICTED". Thus, any finding showing that the owner/admin can steal a user's funds, cause loss of funds or harm to the users, or cause the user's fund to be struck is valid in this audit contest.

> Q: Is the admin/owner of the protocol/contracts TRUSTED or RESTRICTED?
>
> RESTRICTED

The following describes a way where the admin can block users from withdrawing their assets from the protocol

1. The admin calls the `setRebalancer` function to set the rebalance to a wallet address owned by them.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L245

```
File: BaseLSTAdapter.sol
245:     function setRebalancer(address _rebalancer) external onlyOwner {
246:         rebalancer = _rebalancer;
247:     }
```

2. The admin calls the `setTargetBufferPercentage` the set the `targetBufferPercentage` to the smallest possible value of 1%. This will cause only 1% of the total ETH deposited by all the users to reside on the adaptor

contract. This will cause the ETH buffer to deplete quickly and cause all the redemption and withdrawal to revert.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L251

```
File: BaseLSTAdapter.sol
251:     function setTargetBufferPercentage(uint256 _targetBufferPercentage)
 ↪  external onlyRebalancer {
252:         if (_targetBufferPercentage < MIN_BUFFER_PERCENTAGE ||
 ↪  _targetBufferPercentage > BUFFER_PERCENTAGE_PRECISION) {
253:             revert InvalidBufferPercentage();
254:         }
255:         targetBufferPercentage = _targetBufferPercentage;
256:     }
```

3. The owner calls the `setRebalancer` function again and sets the rebalancer address to `address(0)`. As such, no one has the ability to call functions that are only accessible by rebalancer. The `requestWithdrawal` and `requestWithdrawalAll` functions are only accessible by rebalancer. Thus, no one can call these two functions to replenish the ETH buffer in the adaptor contract.
4. When this state is reached, users can no longer withdraw their assets from the protocol, and their assets are stuck in the contract. This effectively causes them to lose their assets.

Impact

Loss of assets for the victim.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L245

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L251

Tool used

Manual Review

Recommendation

To prevent the above scenario, the minimum `targetBufferPercentage` should be set to a higher percentage such as 5 or 10%, and the `requestWithdrawal` function should be made permissionless, so that even if the rebalancer does not do its job, anyone else can still initiate the rebalancing process to replenish the adaptor's ETH buffer for user's withdrawal.

Discussion

sherlock-admin

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

takarez commented:

> invalid

massun-onibakuchi

We are aware of such issues. The owner account is set to governance or multisig. To prevent the target buffer from becoming too low, it is set to 10% by default. This value can be changed even after deployment. Additionally, executing rebalancing functions may reduce the scale of the adapter. Making such functions callable by anyone would, conversely, become a vulnerability. Considering this trade-off, we chose to make it a permissioned function.

nevillehuang

Escalate

Unsure why this issue was excluded. The issue is highlighting how a potentially malicious protocol admin can cause a permanent DoS on users for core functionalities such as redemptions and withdrawals. Given protocol admins are explicitly mentioned as restricted in the contest details, I believe this issue should be valid medium severity.

sherlock-admin2

> Escalate
>
> Unsure why this issue was excluded. The issue is highlighting how a potentially malicious protocol admin can cause a permanent DoS on users for core functionalities such as redemptions and withdrawals. Given protocol admins are explicitly mentioned as restricted in the contest details, I believe this issue should be valid medium severity.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

Agree with Nevi. This report and its duplicates highlighted that it is possible for a malicious admin to negatively impact the users. Thus, it should be valid as per Sherlock's contest rules as admin is "restricted" in the contest's README.

ABDuullahi

This should stay invalid i believe, the README says `Rebalancer: An account can manage adapter and request withdrawal for liquid staking tokens. It can't steal funds.` and this issue does not describe a situation where the role can steal funds.

xiaoming9090

> 4. users can no longer withdraw their assets from the protocol, and their assets are stuck in the contract

The ability for the malicious admin (with rebalancer role) to cause users to be unable to withdraw their assets from the protocol, and in turn lead to their assets being stuck, is sufficient for this issue to be valid.

cvetanovv

I disagree with the escalation. The reason the report is not valid is the first comment from the sponsor. As we can see this is a design decision. Other than that, one of the recommendations is that `targetBufferPercentage` should be higher. But this is a configuration value that can be changed. So I think the report should remain Low/Invalid.

xiaoming9090

> I disagree with the escalation. The reason the report is not valid is the first comment from the sponsor. As we can see this is a design decision. Other than that, one of the recommendations is that `targetBufferPercentage` should be higher. But this is a configuration value that can be changed. So I think the report should remain Low/Invalid.

A malicious admin can set the `targetBufferPercentage` is configured to the lowest possible value, and then calls the `setRebalancer` function again and sets the rebalancer address to address(0). As such, no one has the ability to call functions that are only accessible by rebalancer. The `requestWithdrawal` and `requestWithdrawalAll` functions are only accessible by rebalancer. Thus, no one can call these two functions to replenish the ETH buffer in the adaptor contract.

When the ETH buffer is not replenished, no one can withdraw from the protocol.

This is sufficient to show that it is possible for malicious admins to harm users by preventing them from withdrawing. Note that the admin is restricted in this contest.

ABDuullahi

I think sherlock rules stated that the restriction must be explicitly mentioned, and for this role, its that `it cant steal funds`, not to disrupt the claiming/withdrawal process

nevillehuang

@ABDuullahi This issue is initiated by the admin (owner of contracts), not the rebalancer.

Czar102

> This issue is initiated by the admin (owner of contracts), not the rebalancer.

@nevillehuang is the rebalancer relevant to this issue at all?

If not, I am planning to consider this a Medium severity issue and accept the escalation, with similar reasons to the ones listed here: https://github.com/sherlock-audit/2024-01-napier-judging/issues/97#issuecomment-1997410230.

nevillehuang

@Czar102 Not relevant, the admin can set the rebalancer to an address they control and/or themselves and execute the DoS

Czar102

@nevillehuang @cvetanovv is this a correct and full list of duplicates? #11, #21, #26, #119

Czar102

Result: Medium Has duplicates

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: accepted

Issue M-9: `swapUnderlyingForYt` revert due to rounding issues

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/101

Found by

xiaoming90

Summary

The core function (`swapUnderlyingForYt`) of the Router will revert due to rounding issues. Users who intend to swap underlying assets to YT tokens via the Router will be unable to do so.

Vulnerability Detail

The `swapUnderlyingForYt` allows users to swap underlying assets to a specific number of YT tokens they desire.

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierRouter.sol#L353

```
File: NapierRouter.sol
297:     function swapUnderlyingForYt(
298:         address pool,
299:         uint256 index,
300:         uint256 ytOutDesired,
301:         uint256 underlyingInMax,
302:         address recipient,
303:         uint256 deadline
304:     ) external payable override nonReentrant checkDeadline(deadline)
 ↪  returns (uint256) {
..SNIP..
330:             // Variable Definitions:
331:             // - `uDeposit`: The amount of underlying asset that needs to
 ↪  be deposited to issue PT and YT.
332:             // - `ytOutDesired`: The desired amount of PT and YT to be
 ↪  issued.
333:             // - `cscale`: Current scale of the Tranche.
334:             // - `maxscale`: Maximum scale of the Tranche (denoted as 'S'
 ↪  in the formula).
335:             // - `issuanceFee`: Issuance fee in basis points. (10000 =100%).
336:
337:             // Formula for `Tranche.issue`:
```

```
338:             // ```
339:             // shares = uDeposit / s
340:             // fee = shares * issuanceFeeBps / 10000
341:             // pyIssue = (shares - fee) * S
342:             // ```
343:
344:             // Solving for `uDeposit`:
345:             // ```
346:             // uDeposit = (pyIssue * s / S) / (1 - issuanceFeeBps / 10000)
347:             // ```
348:             // Hack:
349:             // Buffer is added to the denominator.
350:             // This ensures that at least `ytOutDesired` amount of PT and
 ↪  YT are issued.
351:             // If maximum scale and current scale are significantly
 ↪  different or `ytOutDesired` is small, the function might fail.
352:             // Without this buffer, any rounding errors that reduce the
 ↪  issued PT and YT could lead to an insufficient amount of PT to be repaid to
 ↪  the pool.
353:             uint256 uDepositNoFee = cscale * ytOutDesired / maxscale;
354:             uDeposit = uDepositNoFee * MAX_BPS / (MAX_BPS -
 ↪  (series.issuanceFee + 1)); // 0.01 bps buffer
```

Line 353-354 above compute the number of underlying deposits needed to send to the Tranche to issue the amount of YT token the users desired. It attempts to add a buffer of 0.01 bps buffer to prevent rounding errors that could lead to insufficient PT being repaid to the pool and result in a revert. During the audit, it was found that this buffer is ineffective in achieving its purpose.

The following example/POC demonstrates a revert could still occur due to insufficient PT being repaid despite having a buffer:

Let the state be the following:

- cscale = 1.2e18
- maxScale = 1.25e18
- ytOutDesired = 123
- issuanceFee = 0% (For simplicity's sake, the fee is set to zero. Having fee or not does not affect the validity of this issue as this is a math problem)

The following computes the number of underlying assets to be transferred to the Tranche to mint/issue PY + YT

```
uDepositNoFee = cscale * ytOutDesired / maxscale;
uDepositNoFee = 1.2e18 * 123 / 1.25e18 = 118.08 = 118 (Round down)
```


```
uDeposit = uDepositNoFee * MAX_BPS / (MAX_BPS - (series.issuanceFee + 1))
uDeposit = 118 * 10000 / (10000 - (0 + 1)) = 118.0118012 = 118 (Round down)
```

Subsequently, the code will perform a flash-swap via the `swapPtForUnderlying` function. It will borrow 123 PT from the pool, which must be repaid later.

In the swap callback function, the code will transfer 118 underlying assets to the Tranche and execute the `Tranche.issue` function to mint/issue PY + YT.

Within the `Tranche.issue` function, it will trigger the `adapter.prefundedDeposit()` function to mint the estETH/shares. The following is the number of estETH/shares minted:

```
shares = assets * (total supply/total assets)
sahres = 118 * 100e18 / 120e18 = 98.33333333 = 98 shares
```

Next, Line 219 below of the `Tranche.issue` function will compute the number of PY+YT to be issued/minted

```
issued = (sharesUsed - fee).mulWadDown(_maxscale);
issued = (sharesUsed - 0).mulWadDown(_maxscale);
issued = sharesUsed.mulWadDown(_maxscale);

issued = sharesUsed * _maxscale / WAD
issued = 98 * 1.25e18 / 1e18 = 122.5 = 122 PT (Round down)
```

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/Tranche.sol#L219

```
File: Tranche.sol
179:     function issue(
180:         address to,
181:         uint256 underlyingAmount
182:     ) external nonReentrant whenNotPaused notExpired returns (uint256
↪  issued) {
..SNIP..
217:         uint256 sharesUsed = sharesMinted + accruedInTarget;
218:         uint256 fee = sharesUsed.mulDivUp(issuanceFeeBps, MAX_BPS);
219:         issued = (sharesUsed - fee).mulWadDown(_maxscale);
```

At the end of the `Tranche.issue` function, 122 PY + YT is issued/minted back to the Router.

Note that 123 PT was flash-loaned earlier, and 123 PT needs to be repaid. Otherwise, the code at Line 164 below will revert. The main problem is that only 122 PY was issued/minted (a shortfall of 1 PY). Thus, the swap TX will revert at the end.

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierRouter.sol#L164

```
File: NapierRouter.sol
085:     function swapCallback(int256 underlyingDelta, int256 ptDelta, bytes
↪  calldata data) external override {
..SNIP..
161:             uint256 pyIssued = params.pt.issue({to: address(this),
↪  underlyingAmount: params.underlyingDeposit});
162:
163:             // Repay the PT to Napier pool
164:             if (pyIssued < pyDesired) revert
↪  Errors.RouterInsufficientPtRepay();
```

Impact

The core function (swapUnderlyingForYt) of the Router will break. Users who intend to swap underlying assets to YT tokens via the Router will not be able to do so.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/v1-pool/src/NapierRouter.sol#L353

Tool used

Manual Review

Recommendation

The buffer does not appear to be the correct approach to manage this rounding error. One could increase the buffer from 0.01% to 1% and solve the issue in the above example, but a different or larger number might cause a rounding error to surface again. Also, a larger buffer means that many unnecessary PTs will be issued.

Thus, it is recommended that a round-up division be performed when computing the `uDepositNoFee` and `uDeposit` using functions such as `divWadUp` so that the issued/minted PT can cover the debt.

Discussion

sherlock-admin

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

takarez commented:

> valid: rounding error: medium(7)

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/napierfi/v1-pool/pull/158.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-10: Unable to deposit to Tranche/Adaptor under certain conditions

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/105

Found by

AuditorPraise, xiaoming90

Summary

Minting of PT and YT is the core feature of the protocol. Without the ability to mint PT and YT, the protocol would not operate.

The user cannot deposit into the Tranche to issue new PT + YT under certain conditions.

Vulnerability Detail

The comment in Line 133 below mentioned that the `stakeAmount` can be zero.

The reason is that when `targetBufferEth < (availableEth + queueEthCache)`, it is possible that there is a pending withdrawal request (`queueEthCache`) and no available ETH left in the buffer (`availableEth = 0`). Refer to the comment in Line 123 below.

As a result, the code at Line 127 below will restrict the amount of ETH to be staked and set the `stakeAmount` to zero.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L133

```
File: BaseLSTAdapter.sol
071:     function prefundedDeposit() external nonReentrant returns (uint256,
↪  uint256) {
..SNIP..
113:         uint256 stakeAmount;
114:         unchecked {
115:             stakeAmount = availableEth + queueEthCache - targetBufferEth;
↪  // non-zero, no underflow
116:         }
117:         // If the stake amount exceeds 95% of the available ETH, cap the
↪  stake amount.
118:         // This is to prevent the buffer from being completely drained.
↪  This is not a complete solution.
119:         //
```

```
120:         // The condition: stakeAmount > availableEth, is equivalent to:
 ↪  queueEthCache > targetBufferEth
121:         // Possible scenarios:
122:         // - Target buffer percentage was changed to a lower value and
 ↪  there is a large withdrawal request pending.
123:         // - There is a pending withdrawal request and the available ETH
 ↪  are not left in the buffer.
124:         // - There is no pending withdrawal request and the available ETH
 ↪  are not left in the buffer.
125:         uint256 maxStakeAmount = (availableEth * 95) / 100;
126:         if (stakeAmount > maxStakeAmount) {
127:             stakeAmount = maxStakeAmount; // max 95% of the available ETH
128:         }
129:
130:         /// INTERACT ///
131:         // Deposit into the yield source
132:         // Actual amount of ETH spent may be less than the requested amount.
133:         stakeAmount = _stake(stakeAmount); // stake amount can be 0
```

However, the issue is that when `_stake` function is called with `stakeAmount` set to zero, it will result in zero ETH being staked and Line 77 below will revert.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/lido/StEtherAdapter.sol#L77

```
File: StEtherAdapter.sol
64:     /// @inheritdoc BaseLSTAdapter
65:     /// @dev Lido has a limit on the amount of ETH that can be staked.
66:     /// @dev Need to check the current staking limit before staking to
 ↪  prevent DoS.
67:     function _stake(uint256 stakeAmount) internal override returns (uint256)
 ↪  {
68:         uint256 stakeLimit = STETH.getCurrentStakeLimit();
69:         if (stakeAmount > stakeLimit) {
70:             // Cap stake amount
71:             stakeAmount = stakeLimit;
72:         }
73:
74:         IWETH9(Constants.WETH).withdraw(stakeAmount);
75:         uint256 _stETHAmt = STETH.submit{value: stakeAmount}(address(this));
76:
77:         if (_stETHAmt == 0) revert InvariantViolation();
78:         return stakeAmount;
79:     }
```

A similar issue also occurs for the sFRXETH adaptor. If `FRXETH_MINTER.submit`

function is called with `stakeAmount == 0`, it will revert.

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/frax/SFrxETHAdapter.sol#L76

```
File: SFrxETHAdapter.sol
71:     /// @notice Mint sfrxETH using WETH
72:     function _stake(uint256 stakeAmount) internal override returns (uint256)
 ↪  {
73:         IWETH9(Constants.WETH).withdraw(stakeAmount);
74:         FRXETH_MINTER.submit{value: stakeAmount}();
75:         uint256 received = STAKED_FRXETH.deposit(stakeAmount, address(this));
76:         if (received == 0) revert InvariantViolation();
77:
78:         return stakeAmount;
79:     }
```

The following shows that the `FRXETH_MINTER.submit` function will revert if submitted ETH is zero below.

https://etherscan.io/address/0xbAFA44EFE7901E04E39Dad13167D089C559c1138#code#F1#L89

```
/// @notice Mint frxETH to the recipient using sender's funds. Internal portion
function _submit(address recipient) internal nonReentrant {
    // Initial pause and value checks
    require(!submitPaused, "Submit is paused");
    require(msg.value != 0, "Cannot submit 0");
```

Impact

Minting of PT and YT is the core feature of the protocol. Without the ability to mint PT and YT, the protocol would not operate. The user cannot deposit into the Tranche to issue new PT + YT under certain conditions. Breaking of core protocol/contract functionality.

Code Snippet

https://github.com/sherlock-audit/2024-01-napier/blob/main/napier-v1/src/adapters/BaseLSTAdapter.sol#L133

Tool used

Manual Review

Recommendation

Short-circuit the `_stake` function by returning zero value immediately if the `stakeAmount` is zero.

File: StEtherAdapter.sol

```
function _stake(uint256 stakeAmount) internal override returns (uint256) {
+   if (stakeAmount == 0) return 0;
    uint256 stakeLimit = STETH.getCurrentStakeLimit();
    if (stakeAmount > stakeLimit) {
        // Cap stake amount
        stakeAmount = stakeLimit;
    }

    IWETH9(Constants.WETH).withdraw(stakeAmount);
    uint256 _stETHAmt = STETH.submit{value: stakeAmount}(address(this));

    if (_stETHAmt == 0) revert InvariantViolation();
    return stakeAmount;
}
```

File: SFrxETHAdapter.sol

```
function _stake(uint256 stakeAmount) internal override returns (uint256) {
+   if (stakeAmount == 0) return 0;
    IWETH9(Constants.WETH).withdraw(stakeAmount);
    FRXETH_MINTER.submit{value: stakeAmount}();
    uint256 received = STAKED_FRXETH.deposit(stakeAmount, address(this));
    if (received == 0) revert InvariantViolation();

    return stakeAmount;
}
```

Discussion

sherlock-admin3

The protocol team fixed this issue in PR/commit https://github.com/napierfi/napier-v1/pull/169.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-11: FRAX admin can adjust fee rate to harm Napier and its users

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/108

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

FRAX admin can adjust fee rates to harm Napier and its users, preventing Napier users from withdrawing.

Vulnerability Detail

Per the contest page, the admins of the protocols that Napier integrates with are considered "RESTRICTED". This means that any issue related to FRAX's admin action that could negatively affect Napier protocol/users will be considered valid in this audit contest.

> Q: Are the admins of the protocols your contracts integrate with (if any) TRUSTED or RESTRICTED? RESTRICTED

Following is one of the ways that FRAX admin can harm Napier and its users.

FRAX admin can set the fee to 100%.

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3413

```
File: FraxEtherRedemptionQueue.sol
217:     /// @notice Sets the fee for redeeming
218:     /// @param _newFee New redemption fee given in percentage terms, using
 ↪  1e6 precision
219:     function setRedemptionFee(uint64 _newFee) external {
220:         _requireSenderIsTimelock();
221:         if (_newFee > FEE_PRECISION) revert
 ↪  ExceedsMaxRedemptionFee(_newFee, FEE_PRECISION);
222:
223:         emit SetRedemptionFee({ oldRedemptionFee:
 ↪  redemptionQueueState.redemptionFee, newRedemptionFee: _newFee });
224:
```

```
225:         redemptionQueueState.redemptionFee = _newFee;
226:     }
```

When the adaptor attempts to redeem the staked ETH from FRAX via the `enterRedemptionQueue` function, the 100% fee will consume the entire amount of the staked fee, leaving nothing for Napier's adaptor.

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3645

```
File: FraxEtherRedemptionQueue.sol
343:     function enterRedemptionQueue(address _recipient, uint120
↪  _amountToRedeem) public nonReentrant {
344:         // Get queue information
345:         RedemptionQueueState memory _redemptionQueueState =
↪  redemptionQueueState;
346:         RedemptionQueueAccounting memory _redemptionQueueAccounting =
↪  redemptionQueueAccounting;
347:
348:         // Calculations: redemption fee
349:         uint120 _redemptionFeeAmount = ((uint256(_amountToRedeem) *
↪  _redemptionQueueState.redemptionFee) /
350:             FEE_PRECISION).toUint120();
351:
352:         // Calculations: amount of ETH owed to the user
353:         uint120 _amountEtherOwedToUser = _amountToRedeem -
↪  _redemptionFeeAmount;
354:
355:         // Calculations: increment ether liabilities by the amount of ether
↪  owed to the user
356:         _redemptionQueueAccounting.etherLiabilities +=
↪  uint128(_amountEtherOwedToUser);
357:
358:         // Calculations: increment unclaimed fees by the redemption fee
↪  taken
359:         _redemptionQueueAccounting.unclaimedFees += _redemptionFeeAmount;
360:
361:         // Calculations: maturity timestamp
362:         uint64 _maturityTimestamp = uint64(block.timestamp) +
↪  _redemptionQueueState.queueLengthSecs;
363:
364:         // Effects: Initialize the redemption ticket NFT information
365:         nftInformation[_redemptionQueueState.nextNftId] =
↪  RedemptionQueueItem({
366:             amount: _amountEtherOwedToUser,
367:             maturity: _maturityTimestamp,
368:             hasBeenRedeemed: false,
```

```
369:             earlyExitFee: _redemptionQueueState.earlyExitFee
370:         });
```

Impact

Users unable to withdraw their assets. Loss of assets for the victim.

Code Snippet

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3413

https://etherscan.io/address/0x82bA8da44Cd5261762e629dd5c605b17715727bd#code#L3645

Tool used

Manual Review

Recommendation

Ensure that the protocol team and its users are aware of the risks of such an event and develop a contingency plan to manage it.

Discussion

sherlock-admin

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

takarez commented:

> valid: medium(11)

Issue M-12: The pool verification in `NapierRouter` is prone to collision attacks

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/111

The protocol has acknowledged this issue.

Found by

Arabadzhiev

Summary

Using computed pool addresses to verify the callback function callers is collision attack prone and can be abused in order to steal all token allowances of the `NapierRouter` contract

Vulnerability Detail

With the current state of the `NapierRouter` contract implementation, both the `mintCallback` and `swapCallback` use the `_verifyCallback` function in order to verify that the address that called them is a Napier pool deployed by the `PoolFactory` attached to the router. This is done by computing the Create2 pool address using the `basePool` and `underlying` values passed in to either of the callbacks as arguments and the comparing that address to the `msg.sender`.

However, this method of verifying the caller address is collision prone, as the computation of the Create2 address is done by truncating a 256 bit keccak256 hash to 160 bits, meaning that for each address there are 2^96 possible hashes that will result in it after being truncated. Furthermore, what this means is that if a `basePool` and `underlying` combination that results in an address controlled by a malicious user is found, all token allowances given to the `NapierRouter` contract can be stolen.

In this article and this report from a past Sherlock contest, it is shown that this vulnerability will likely cost a few million dollars as of today, but due to the rapid advancement in computing capabilities, it is likely that it will cost much less a few years down the line.

Impact

All token allowances of the `NapierRouter` can be stolen

Code Snippet

NapierRouter.sol#L65-L68

Tool used

Manual Review

Recommendation

In the `PoolFactory` contract, create a function that verifies that a given pool address has been deployed by it using the `_pools` mapping. Example implementation:

```
contract PoolFactory is IPoolFactory, Ownable2Step {
    ...
    /// @notice Mapping of NapierPool to PoolAssets
    mapping(address => PoolAssets) internal _pools;
    ...
+   function isFactoryDeployedPool(address poolAddress) external view returns
↪   (bool) {
+       PoolAssets memory poolAssets = _pools[poolAddress];
+       return (
+           poolAssets.basePool != address(0) &&
+           poolAssets.underlying != address(0) &&
+           poolAssets.principalTokens.length != 0
+       );
    }
}
```

And then use it in the `NapierRouter::_verifyCallback` function in place of the computed address comparison logic:

```
    function _verifyCallback(address basePool, address underlying) internal view
↪   {
-       if (
-           PoolAddress.computeAddress(basePool, underlying, POOL_CREATION_HASH,
↪   address(factory))
-               != INapierPool(msg.sender)
-       ) revert Errors.RouterCallbackNotNapierPool();
+       if (factory.isFactoryDeployedPool(msg.sender)) revert
↪   Errors.RouterCallbackNotNapierPool();
    }
```

Discussion

sherlock-admin4

> Escalate
>
> Invalid. Both the mentioned report requires only 2^80 computation to find a collision hash because of the birthday paradox. In this issue, the attacker needs approx 2^160 computation to find the exact match with the specific napier pool deployed.

You've deleted an escalation for this issue.

Arabadzhiew

> Escalate
>
> Invalid. Both the mentioned report requires only 2^80 computation to find a collision hash because of the birthday paradox. In this issue, the attacker needs approx 2^160 computation to find the exact match with the specific napier pool deployed.

How exactly did you come to that conclusion? The goal of this exploit is not to find a match with a specific Napier pool, but instead a match with an EOA controlled by the attacker. The case is the same in the mentioned report. If anything, in the case of the `NapierRouter` the collision would be theoretically more likely to happen, since here we have two address arguments (320 bits) that can be played with when trying to find collision, while in the Kyberswap issue there is an address + uint24 (184 bits).

Coareal

> How exactly did you come to that conclusion? The goal of this exploit is not to find a match with a specific Napier pool, but instead a match with an EOA controlled by the attacker. The case is the same in the mentioned report. If anything, in the case of the `NapierRouter` the collision would be theoretically more likely to happen, since here we have two address arguments (320 bits) that can be played with when trying to find collision, while in the Kyberswap issue there is an address + uint24 (184 bits).

Removed escalation. Agree that issue is similar to Kyberswap's.

nevillehuang

Escalate

How would the allowances be stolen here? Wouldn’t a user be only approving specified PT amounts wherein every swap will consume the approval, thus making this attack not profitable?

Additionally, This issue seems to lack a further description/explanation of the attack vector specific to napier as opposed to the linked issue:

> token0 can be constant and we can achieve the variation in the hash by changing token1. The attacker could use token0 = WETH and vary token1. This would allow them to steal all allowances of WETH.

Perhaps the watsons @Arabadzhiew @xiaoming9090 can clarify better

sherlock-admin2

> Escalate
>
> How would the allowances be stolen here? Wouldn’t a user be only approving specified PT amounts wherein every swap will consume the approval, thus making this attack not profitable?
>
> Additionally, This issue seems to lack a further description/explanation of the attack vector specific to napier as opposed to the linked issue:
>
> > token0 can be constant and we can achieve the variation in the hash by changing token1. The attacker could use token0 = WETH and vary token1. This would allow them to steal all allowances of WETH.
>
> Perhaps the watsons @Arabadzhiew @xiaoming9090 can clarify better

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Arabadzhiew

> Escalate
>
> How would the allowances be stolen here? Wouldn’t a user be only approving specified PT amounts wherein every swap will consume the approval, thus making this attack not profitable?
>
> Additionally, This issue seems to lack a further description/explanation of the attack vector specific to napier as opposed to the linked issue:
>
> > token0 can be constant and we can achieve the variation in the hash by changing token1. The attacker could use token0 = WETH and vary token1. This would allow them to steal all allowances of WETH.
>
> Perhaps the watsons @Arabadzhiew @xiaoming9090 can clarify better

Most of the users that will perform more than 1 swap with a specific token using the router will simply give max approval to it. This is because:

1. It saves gas by not having to make a separate approve call for each swap;
2. By default, this is what you are prompted to do by Metamask when you are required to give token allowance to some dapp;

Additionally, I have only given a high level overview of the issue, since it is very similar to the one reported in the Kyberswap contest, and I thought that a more in-depth explanation would simply be redundant. But since you are asking about it, I'm going to provide a short explanation on one of the ways of how this vulnerability can be exploited in the context of Napier:

Taking the `NapierRouter::swapCallback` function:

```
function swapCallback(int256 underlyingDelta, int256 ptDelta, bytes calldata
↪  data) external override {
    // `data` is encoded as follows:
    // [0x00: 0x20] CallbackType (uint8)
    // [0x20: 0x40] Underlying (address)
    // [0x40: 0x60] BasePool (address)
    // [0x60:  ~  ] Custom data (based on CallbackType)
    (address underlying, address basePool) = abi.decode(data[0x20:0x60],
↪  (address, address));
    _verifyCallback(basePool, underlying);

    CallbackType _type = CallbackDataTypes.getCallbackType(data);

    if (_type == CallbackType.SwapPtForUnderlying) {
        CallbackDataTypes.SwapPtForUnderlyingData memory params =
            abi.decode(data[0x60:], (CallbackDataTypes.SwapPtForUnderlyingData));
        params.pt.safeTransferFrom(params.payer, msg.sender, uint256(-ptDelta));
    ...
}
```

As it can be seen in this snippet, the `basePool` and `underlying` values are only used for the pool address verification. Furthermore, we can see that in the first `if` block of the function there is a single `safeTransferFrom` call that sends a concrete **amount** of a concrete **token** from a concrete **address** to the `msg.sender`. All the values in bold are passed in as function arguments and are not used for the CREATE2 pool address computation. What we can conclude from this is that if someone manages to find a combination of `basePool` and `underlying` that passes the `_verifyCallback` check for their EOA, then they can easily steal all of the token allowances given to the router contract.

0xMRO

`msg.sender` is already being validated as napier pool address.

```
/// @dev Revert if `msg.sender` is not a Napier pool.
function _verifyCallback(address basePool, address underlying) internal view {
    if (
        PoolAddress.computeAddress(basePool, underlying, POOL_CREATION_HASH,
↪   address(factory))
            != INapierPool(msg.sender)
    ) revert Errors.RouterCallbackNotNapierPool();
}
```

It would revert for non-existent napier pool address. How the recommendation is different from the above check?

Arabadzhiew

@0xMR0 Quoting what is stated in my report:

> However, this method of verifying the caller address is collision prone, as the computation of the Create2 address is done by truncating a 256 bit keccak256 hash to 160 bits, meaning that for each address there are 2^96 possible hashes that will result in it after being truncated. Furthermore, what this means is that if a basePool and underlying combination that results in an address controlled by a malicious user is found, all token allowances given to the NapierRouter contract can be stolen.

Additionally, the recommendation is different from the current approach of verification in that, it will verify whether the `msg.sender` is a legit Nepier pool by checking that it was deployed by the `PoolFactory` rather than comparing it to a computed Create2 address, which is collision prone.

Please read the full report and the resources attached to it in order to gain a deeper understanding of the vulnerability in question.

massun-onibakuchi

I think CREATE2 with Factory pattern can be found in well-known UniswapV3, and I believe Uniswap v3 don't need to fix it because the issue seems to be mostly impossible to happen in reality. Therefore, this issue should be considered as a low or non-issue. References: https://github.com/Uniswap/v3-periphery/blob/main/contracts/SwapRouter.sol#L65 https://github.com/Uniswap/v3-periphery/blob/main/contracts/libraries/CallbackValidation.sol#L21

Also, the attack cost makes a strong external condition, which requires a very big upfront cost with no certainty on pay out as attacker.

nevillehuang

To keep consistency of results, I think this issue should remain valid, but I am keeping my escalation up for @Czar102 to revisit this type of issues. I believe based on sponsor comments here and the assumption that all users make max approvals towards the router, this type of issues should have a better sherlock rule catering to this and/or make known to sponsors to be aware as a known accepted risk.

Arabadzhiew

The reason why such a vulnerability has not been exploited in the wild yet is that as @massun-onibakuchi mentioned, the attack requires a big upfront cost as of today. However, given the fact that `NapierRouter` is an immutable contract and taking into consideration the rapid advancement of computing capabilities (the Bitcoin hashrate alone has almost doubled since the Kyberswap issue was reported - it has raised from 4.71e20 to 7.68e20), we can see how this exploit might become much more profitable and very realistically feasible in the not so distant future. And as of now, the Sherlock rulebook does not have a fixed timeline within which an issue must occur, in order for it to be considered valid.

Also, the allowances that are going to be given to the contract are not an assumption, but rather a "fact of life" for such kind of contracts. Pretty much all DEX router contracts have token allowances in the magnitudes of millions of dollars. And that is because understandably, nobody wants to pay extra gas fees when possible.

On a final note, in the particular case of the `NapierRouter` **all** token allowances given to it can be stolen with a single address collision, while in the case of the Kyberswap issue, a separate collision has to be found for each token. That is because the CREATE2 addresses of the Napier pools are not computed based on all token addresses used within them.

I will refrain from commenting on this issue any further. I believe that enough has been said in order for the Head of Judging to be able to make an informed decision.

xiaoming9090

A note to the judges. The risk profile of this issue is the same as the Arcadia's Contest (https://github.com/sherlock-audit/2023-12-arcadia-judging/issues/59). Thus, the risk rating of this issue should be aligned for consistency. Thanks.

cvetanovv

This issue should remain Medium. The report is similar to the Arcadia contest from a few weeks ago. It wouldn't be fair for it to be Low here. However, I suggest that Sherlock include it in the documentation as a known issue in the future.

Czar102

To exploit this, an attacker would need to find a collision between an owned address and a potentially deployed pool (if there is less than 10^12 pools deployed

then the attack is definitely unfeasible). So, the attacker needs to make the owner deploy a pool. The attacker needs to be the owner.

The issue by itself is a borderline Med/Low, so I believe, considering this additional constraint needed to execute it, this issue should be of Low severity.

Please correct me if I'm understanding something wrongly.

Arabadzhiew

@Czar102 There doesn't have to be a deployed pool at the given address. The goal of the exploit is to find a combination of the `basePool` and `underlying` values that results in an address that is equal to an EOA controlled by the malicious user. Then, that user will be able to call any one of the two callbacks through that EOA and steal the token allowances given to the router contract.

Czar102

@Arabadzhiew indeed, thank you for correcting me!

I'm planning to reject the escalation and leave this a valid Medium severity issue.

Czar102

On second thoughts, @Arabadzhiew can you elaborate how much computation needs to be done having ~2^50 memory available?

If an algorithm for finding this collision is not provided, I'm planning to invalidate this finding, similarly to findings that do not describe the algorithm in the future. (full feasibility analysis should have been done)

Arabadzhiew

Using the so-called "naive" collision finding algorithm, which is the one in question in all of the resources I have attached to my report requires at least 2^80 of storage available. And because that is an enormous amount as of today's standards, this is the biggest bottleneck of this exploit. What this essentially means is that with 2^50 of memory, the exploit will be practically impossible to execute using this algorithm.

To show how much more computation will have to be made with this amount of memory, let's look at some math. I'm going to use the following formula to calculate the hash collision probabilities:

$$\frac{k^2}{2N}$$

Where **k** is the number of computed hashes and **N** is the total number of possible hashes (2^160 in our case).

You can read more about it here, but essentially, it is a very simplified version of a more complex hash collision probability formula, that works well with very big

numbers. This is the least accurate formula from all of the mentioned ones in the article, but it is the only one that will return a probability value that is different from 0 with such a low **k** value being passed in to it.

So for the scenario where we have 2^50 of memory available, we will first generate 2^50 public addresses, store those in our memory and then compute another set of 2^50 different pool addresses (by using different `basePool` and `underlying` values for each one). In that case, we will have **k = 2^51**. The output that we will get for this value from the above formula is the following: **1.7347235e-18** - a very small decimal value. This is our probability value between 0 and 1, where 0 = 0% probability and 1 = 100% probability. So to get how many times we will have to make 2^50 computations in order to have ~100% probability of collision, we have to divide 1 by that value. This results in the following value: **5.7646075e+17** which is exactly equal to 2^59. What we can derive from that is that to have approximately 100% probability of finding a collision in a 160 bit space, with only 2^50 of memory available, we will have to perform **2^50 * 2^59 = 2^109** computations. Obviously, this is a massive amount of computations. To put into perspective how big it actually is, if we had the full Bitcoin computation power at the hashrate of 7.68e20, it would take us **26797.9 years** to make this many computations.

On the other hand though, if we had 2^80 of memory available, then the probability values will look completely differently. We will perform the exact same procedure as above, but this time we will generate two sets of 2^80 addresses. So our value of **k** will be equal to **2^81** this time around. And because that value is much bigger than 2^51, it allows us to use the other more accurate formulas. I will be using the following formula for this probability calculation, which is simply a more accurate version of the one used above:

$$1 - e^{\frac{-k(k-1)}{2N}}$$

The output that we get from that formula is **0.86466471676** which represents **~86.46%** chance of collision for the computation of 2^81 addresses. That is much better than than the previous example. And if we bump up our computations to 2^82, then we will get **0.99966453737** as an output, which represents **~99.96** chance of collision for that many computations. And for the sake of comparison, 2^81 computations can be performed in **0.87** hours by the Bitcoin network at the hashrate of 7.68e20 and 2^82 in **1.74** hours.

On a final note, I'd like to mention once again that this is the simplest form of a hash collision finding algorithm, which does not make use of any memory saving optimizations. There are potential ways to bring down the amount of memory required for performing it, such as using bloom filters as mentioned in this comment under the Kyberswap issue, so I'm sure that with time we will see more and more feasible implementations of that algorithm.

@Czar102 I hope this clarifies things further.

Czar102

> On a final note, I'd like to mention once again that this is the simplest form of a hash collision finding algorithm, which does not make use of any memory saving optimizations. There are potential ways to bring down the amount of memory required for performing it, such as using bloom filters as mentioned in this comment under the Kyberswap issue, so I'm sure that with time we will see more and more feasible implementations of that algorithm.

Unless these optimizations are shared, I believe address collision to be a non-issue. Bloom filter can optimize storage down to 1 bit, but I don't think it's possible to go much further based on information-theoretic limitations: when a function $`f : \mathbb{B}^{20} \to \{0,1\}`$ (describing whether a hash was already yielded) has at most $2^{2^{50}}$ configurations, the "expected value of information" about any random hash can't be larger than 2^{-110}, even with fractional information (probabilistic approaches).

Of course, my rough understanding may be wrong, so I'm open to having my mind changed. For now, this attack seems to be impossible to carry out. Hence, I'm planning to invalidate this finding.

midori-fuse

As the submitter of the same issue in Arcadia, this conversation has been quite helpful, and we'd like to share our thoughts.

When we investigated the issue in Arcadia, we only really thought of the time complexity of $O(2^{N/2})$, which is possible, but not the memory complexity, which is indeed impossible as of now. The head of judging's question made us realize the additional constraint, and we apologize for not realizing it earlier during the Arcadia contest.

However, as the head of judging stated himself in the Sherlock discord, I do believe the continuous development of available resources and what's at stake makes this issue worth mitigating. This is the kind of issue that's of critical impact if were to happen, and while it's impossible as of now, there's no telling *when* it will be possible in the future.

Hence, while I believe this issue is true, I don't have opinions about its severity. What I would consider is what are the odds it becomes realistic in the lifetime of this protocol, not just at this current moment.

- In other words we're betting on how much can the storage problem can be solved, and how much funds are still at stake by then. This is also one of the discussion points from the Kyber issue.

I do believe, however, that it's the responsibility of the issue author to outline the

constraints related to the attack, and these kind of issues will likely need a revisit in the future given technological advancements.

Arabadzhiew

After doing some further research on the topic, I came across this paper, which describes a much more sophisticated collision finding algorithm. It requires 160 Terabytes (~2^47) of memory and 2^92 computations. Now, in my eyes, this looks like a much more feasible implementation. The number of computations is quite a bit higher, but it is still well within the realm of feasibility. Using the analogy that I've used above, the Bitcoin network will need ~0.2 years at the hashrate of 7.68e20 to make that many computations.

The thing is, if someone decides to try and perform this exploit in the future, they won't be targeting a single protocol. They will be targeting as much protocols as they possibly can. So the more protocols there are with this vulnerability in them, the more likely it will be that this gets exploited.

Czar102

> The thing is, if someone decides to try and perform this exploit in the future, they won't be targeting a single protocol. They will be targeting as much protocols as they possibly can. So the more protocols there are with this vulnerability in them, the more likely it will be that this gets exploited.

I don't think that's true given the memory complexity of this attack.

Nevertheless, the linked paper does present a way to optimize the memory complexity beyond the approach outlined in the previous comments. A large (2^{11}) multiplier in the above example is due to the increased difficulty of computing the elliptic point operations, which isn't needed in this attack – the attacker can simply deploy a wallet controlled by them using CREATE2. This will make the computational complexity of around 2^{81} and would require giga-terabytes of memory, which means this attack is possible to execute with ~34 minutes of the current Bitcoin's hashrate, assuming keccak256 is as easy to calculate as SHA-2. The current cost of this attack would be less than $1.5m with the current prices. This, of course, doesn't take into account the fact that keccak256 mining is a niche field, so the extent of cost optimization is much worse.

I'd also like @midori-fuse to sign off on my reasoning – would you agree that it's possible to execute this attack?

I think this finding was correctly judged as a Medium severity one.

midori-fuse

This is a really cool algorithm. Massive kudos to @Arabadzhiew for finding this paper.

Indeed that since we're looking for a contract-contract collision and not an EOA-contract collision, no EC multiplication is needed, making f_1 and f_2 purely keccak256 just with different inputs, and the cost of the attack is about 2^{81} calls to f (which is, again, just keccak256).

Since both functions are of the same weight now, we can set $m = \frac{1}{\theta} = 2^{40}$ and achieve the same execution time, but with ~ 16 times lower processor count and memory cost than outlined. Of course, the real attack time would take longer depending on how many parallel processes you have, but taking 0.02% of this power for 5000x the duration (while also lowering the memory cost by 5000x) and it's still not unrealistically long. The total number of hashing calls across all processors is the same, so the attack cost is the same as outlined.

Under the assumption that a standard computer can perform 2^{30} operations per second which is reasonable, @Czar102 's calculation is on point. Taking technological advancements into consideration, the computing power here does not have to be solely keccak256 mining, but also overall processor improvements as well.

nevillehuang

Maybe off topic and not relevant to finding but Curious why nobody tried to exploit uniswapv3

Czar102

@nevillehuang probably because you would need to create your own ASIC just for this (millions) or pay a quarter billion dollars for many, many GPU-hours.

nevillehuang

@Czar102 Surely its worth it for say a 400+ million USDC/ETH univ3 pool right?

Czar102

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: rejected

Issue M-13: SFrxETHAdapter redemptionQueue waiting period can DOS adapter functions

Source: https://github.com/sherlock-audit/2024-01-napier-judging/issues/120

The protocol has acknowledged this issue.

Found by

Falconhoof, mahdikarimi, xiaoming90

Links

https://github.com/FraxFinance/frax-ether-redemption-queue/blob/17ebebcddf31b7780e92c23a6b440dc789e5ceac/src/contracts/FraxEtherRedemptionQueue.sol#L417-L461 https://github.com/FraxFinance/frax-ether-redemption-queue/blob/17ebebcddf31b7780e92c23a6b440dc789e5ceac/src/contracts/FraxEtherRedemptionQueue.sol#L235-L246
https://github.com/sherlock-audit/2024-01-napier/blob/6313f34110b0d12677b389f0ecb3197038211e12/napier-v1/src/adapters/BaseLSTAdapter.sol#L71-L139
https://github.com/sherlock-audit/2024-01-napier/blob/6313f34110b0d12677b389f0ecb3197038211e12/napier-v1/src/adapters/BaseLSTAdapter.sol#L146-L169

Summary

The waiting period between `rebalancer` address making a withdrawal request and the withdrawn funds being ready to claim from `FraxEtherRedemptionQueue` is extremely long which can lead to a significant period of time where some of the protocol's functions are either unusable or work in a diminished capacity.

Vulnerability Detail

In `FraxEtherRedemptionQueue.sol`; the Queue wait time is stored in the state struct `redemptionQueueState` as `redemptionQueueState.queueLengthSecs` and is curently set to `1_296_000 Seconds` or `15 Days`; as recently as January however it was at `1_555_200 Seconds` or `18 Days`. View current setting by calling `redemptionQueueState()` here.

`BaseLSTAdapter::requestWithdrawal()` is an essential function which helps to maintain `bufferEth` at a defined, healthy level. `bufferEth` is a facility which smooth running of redemptions and deposits.

For `redemptions`; it allows users to redeem `underlying` without having to wait for any period of time. However, redemption amounts requested which are less than `bufferEth` will be rejected as can be seen below in

`BaseLSTAdapter::prefundedRedeem()`. Further, there is nothing preventing `redemptions` from bringing `bufferEth` all the way to `0`.

```
    function prefundedRedeem(address recipient) external virtual returns
↪   (uint256, uint256) {
        // SOME CODE

        // If the buffer is insufficient, shares cannot be redeemed immediately
        // Need to wait for the withdrawal to be completed and the buffer to be
↪   refilled.
>>>     if (assets > bufferEthCache) revert InsufficientBuffer();

        // SOME CODE
    }
```

For `deposits`; where `bufferEth` is too low, it keeps user deposits in the contract until a deposit is made which brings `bufferEth` above it's target, at which point it stakes. During this time, the deposits, which are kept in the adapter, do not earn any yield; making those funds unprofitable.

```
    function prefundedDeposit() external nonReentrant returns (uint256, uint256)
↪   {
    // SOME CODE
>>>     if (targetBufferEth >= availableEth + queueEthCache) {
>>>         bufferEth = availableEth.toUint128();
            return (assets, shares);
        }
    // SOME CODE
    }
```

Impact

If the `SFrxETHAdapter` experiences a large net redemption, bringing `bufferEth` significantly below `targetBufferEth`, the rebalancer can be required to make a withdrawal request in order to replenish the buffer. However, this will be an ineffective action given the current, 15 Day waiting period. During the waiting period if `redemptions > deposits`, the bufferEth can be brought down to `0` which will mean a complete DOSing of the `prefundedRedeem()` function.

During the wait period too; if `redemptions >= deposits`, no new funds will be staked in `FRAX` so yields for users will decrease and may in turn lead to more redemptions.

These conditions could also necessitate the immediate calling again of `requestWithdrawal()`, given that withdrawal requests can only bring `bufferEth` up to it's target level and not beyond and during the wait period there could be further

redemptions.

Code Snippet

Simple example with Yield on `sFrxETHBalance` ignored:

Start off with 100 wETH deposited; 10 wETH `bufferEth`

> totalAssets() = (withdrawalQueueEth + bufferEth + sFrxETHBalance)
> totalAssets() = (0 + 10 + 90)

Net Redemption 5 wETH reduces bufferEth so rebalancer makes Withdrawl Request of 4.5 wETH to bring bufferEth to 10% (9.5 wEth)

> totalAssets() = (4.5 + 5 + 85.5)

During the wait period, continued Net Redemption reduces bufferEth further requiring another withdrawl request by rebalancer for 4.05 wEth

> totalAssets() = (0 + 4.5 + 85.5)

Tool used

Foundry Testing Manual Review

Recommendation

Consider adding a function allowing the rebalancer call `earlyBurnRedemptionTicketNft()` in `FraxEtherRedemptionQueue.sol` when there is a necessity. This will allow an immediate withdrawal for a fee of `0.5%`; see function here

Discussion

massun-onibakuchi

We are aware of the situation. Therefore, we plan to set the TARGET BUFFER PERCENTAGE passively.

nevillehuang

Escalate, this is a duplicate of #89 given it shares the exact same root cause of depletion of eth buffer to cause dos in withdrawals

sherlock-admin2

> Escalate, this is a duplicate of #89 given it shares the exact same root cause of depletion of eth buffer to cause dos in withdrawals

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Qormatic

Malicious intent is not required for this to be an issue; which it will be as it will DOS user's ability to access funds for an extended period of time. As noted in the issue; the withdrawal design aims to enable users withdraw their funds from Napier immediately without having to wait in the LST protocols withdrawal queue so this issue breaks the design of the protocol.

> For redemptions; it allows users to redeem underlying without having to wait for any period of time.

Further; as per Sherlock docs; a DOS can be considered valid if users funds are locked up for more than a week. Given that, in this case, users funds would be locked up for 15 days it more than meets that threshold.

> Could Denial-of-Service (DOS), griefing, or locking of contracts count as a Medium (or High) issue? DoS has two separate scores on which it can become an issue: The issue causes locking of funds for users for more than a week.

nevillehuang

@Qormatic You are correct, adjusted my escalation.

massun-onibakuchi

@nevillehuang it is expected that most of users will basically withdraw after the maturity because redemption of principal token can be only allowed after the maturity. (`redemptions` » `deposits`: after the maturity. `redemptions` « `deposits`: before the maturity.). so, considering the redemption queue time, we're going to set targetBufferPercentage to higher one over time and before `bufferEth` derecase or 2 weeks before the the maturity, we will request withdrawal because our team knows it'd take 2 weeks to unstake ETH before this audit. The DoS can be caused by mismanagement. We believe that using waiting time as the basis for the claim does not reflect the actual situation.

massun-onibakuchi

> Malicious intent is not required for this to be an issue; which it will be as it will DOS user's ability to access funds for an extended period of time.

I think this finding needs unlikely assumption. To borrow a phrase, this issue requires malicious intent.

Qormatic

@massun-onibakuchi I don't think it's an unlikely assumption that users would want to withdraw funds before maturity; else why would you provide them with zero wait functionality to do so?

And the Admin team may have a plan to manage a best case scenario but the DOS vulnerability would be caused by user actions outside the admin team's control and would negatively impact other users by locking up their funds for an extended period of time.

Also i dont think its fair post-contest to introduce additional context about off-chain withdrawl & buffer management which wasn't included in the README.

massun-onibakuchi

@Qormatic why do you think it's not unlikely? From economical perspective, it is easily expected that above. Actually after the maturity deposits will be reverted by contract and only after the maturity, redemption of PT is allowed.

`targetBufferPercentage` is state variable can be changed by `setTargetBufferPerecntage()`, which intends the value can be adjusted properly after deployment. That's why it has setter function and not immutable.

Qormatic

@massun-onibakuchi why can't user call redeemWithYT(); there's no expired modifier on it?

massun-onibakuchi

> @massun-onibakuchi why can't user call redeemWithYT(); there's no expired modifier on it?

@Qormatic This method requires users to holds the same amount of PT/YT.

Banditx0x

This is clearly the design of the protocol. It is impossible to always be able to redeem all tokens from a protocol (like Napier) which stakes tokens into a staking protocol with a redemption queue/delay.

Napier is designed to reduce the likelihood of a withdrawal being delayed, but cannot permanantly remove it.

xiaoming9090

> @nevillehuang it is expected that most of users will basically withdraw after the maturity because redemption of principal token can be only allowed after the maturity. (`redemptions` » `deposits`: after the maturity. `redemptions` « `deposits`: before the maturity.). so, considering the redemption queue time, we're going to set targetBufferPercentage to

> higher one over time and before `bufferEth` derecase or 2 weeks before the the maturity, we will request withdrawal because our team knows it'd take 2 weeks to unstake ETH before this audit. The DoS can be caused by mismanagement. We believe that using waiting time as the basis for the claim does not reflect the actual situation.

The protocol is designed to allow users to withdraw both before and after maturity. Before maturity, users can withdraw via the `redeemWithYT` function. After maturity, the users can withdraw via the `redeem` or `withdraw` function. The report and its duplicate have shown that it is possible for malicious actors to DOS the user withdrawal, which is quite severe.

The above measures do not fully mitigate the issue. If the maturity period is a year, the `targetBufferPercentage` will only be set to a high value when it is near maturity.

Thus, the `targetBufferPercentage` has to be low at all other times during the one-year period. Otherwise, the protocol's core earning mechanism is broken. So, the malicious actor could still carry out the attack for the vast majority of the one-year period, resulting in users being unable to withdraw.

The approach to fully mitigate this issue is documented in my report (https://github.com/sherlock-audit/2024-01-napier-judging/issues/89).

xiaoming9090

> This is clearly the design of the protocol. It is impossible to always be able to redeem all tokens from a protocol (like Napier) which stakes tokens into a staking protocol with a redemption queue/delay.
>
> Napier is designed to reduce the likelihood of a withdrawal being delayed, but cannot permanantly remove it.

If appropriate fees and restrictions had been put in place (See the recommendation section of https://github.com/sherlock-audit/2024-01-napier-judging/issues/89), this issue would not have occurred in the first place. Thus, this is not related to the design of the protocol.

xiaoming9090

> @massun-onibakuchi I don't think it's an unlikely assumption that users would want to withdraw funds before maturity; else why would you provide them with zero wait functionality to do so?
>
> And the Admin team may have a plan to manage a best case scenario but the DOS vulnerability would be caused by user actions outside the admin team's control and would negatively impact other users by locking up their funds for an extended period of time.
>
> Also i dont think its fair post-contest to introduce additional context about off-chain withdrawl & buffer management which wasn't included in

the README.

Agree with this. Note that this DOS issue, which is quite severe, is not documented under the list of known issues of the Contest's README. Thus, it would only be fair for Watson to flag this issue during the contest. Also, the mitigation mentioned by the sponsor cannot be applied retrospectively after the contest.

cvetanovv

I agree with @nevillehuang escalation. This issue can be a dup of 89. There I have left a comment on what I think of the report.

Czar102

I agree with the escalation. Planning to consider this a duplicate of #89.

Czar102

During the escalation period this issue was valid and #89 – invalid. Hence, the duplication of #89 <> #120 should be proposed on #89. There already was an escalation there that showed that the issue is valid, so there is only a single mistake made here – #89 is not a duplicate of #120, but invalid. Hence, one escalation should be accepted and it was the one on #89. This issue is not undergoing any changes.

Hence, planning to apply the suggestion to duplicate these two, but also reject the escalation here.

nevillehuang

@Czar102 So this is a dupe of #89? Applying changes but reject escalation correct?

Czar102

Yes, I edited my comment to be clearer. Technically, will consider #89 to be a duplicate of this issue.

Czar102

Result: Medium Has duplicates

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: rejected

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.