SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	Sophon
Prepared by:	Sherlock
Lead Security Expert:	zzykxx
Dates Audited:	May 20 - May 24, 2024
Prepared on:	June 17, 2024

Introduction

Sophon, an entertainment-focused hyperchain on ZkSync, is hosting an airdrop process. In this process, anyone can stake certain assets to qualify for a $SOPHairdropscheduledforQ3, 2024.$

Scope

Repository: sophon-org/farming-contracts

Branch: main

Commit: c08a2a81f2d5359bc05f41a25df54c7a91a73a03

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
3	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

zzykxx	serial-coder	araj
EgisSecurity	blackhole	utsav
Bauchibred	MightyRaju	0xAadi
ZdravkoHr.	yamato	dhank
h2134	jecikpo	underdog
hunter_w3b	KupiaSec	
p0wd3r	whitehair0330	

Issue H-1: The quantity is calculated incorrectly when depositing ETH to weETH.

Source: https://github.com/sherlock-audit/2024-05-sophon-judging/issues/4

Found by

EgisSecurity, hunter_w3b, p0wd3r, zzykxx

Summary

The quantity is calculated incorrectly when depositing ETH to weETH.

The code treats **the quantity of eETH shares** returned by Etherfi `LiquidityPool.deposit` as **the actual quantity of eETH**, but these two quantities are not equal.

The Etherfi `LiquidityPool.deposit` and `stETH.submit` functions have the same behavior, both returning shares instead of the actual token amount. The protocol handles stETH correctly, but it doesn't handle eETH correctly.

Vulnerability Detail

In `depositEth`, if `_predefinedPool == PredefinedPool.weETH`, `_ethTOeEth` will be called to get the `finalAmount`.

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L503-L516

```
function depositEth(uint256 _boostAmount, PredefinedPool _predefinedPool) public
↪  payable {
    if (msg.value == 0) {
        revert NoEthSent();
    }

    uint256 _finalAmount = msg.value;
    if (_predefinedPool == PredefinedPool.wstETH) {
        _finalAmount = _ethTOstEth(_finalAmount);
    } else if (_predefinedPool == PredefinedPool.weETH) {
        _finalAmount = _ethTOeEth(_finalAmount);
    }

    _depositPredefinedAsset(_finalAmount, msg.value, _boostAmount,
↪  _predefinedPool);
}
```

`_ethTOeEth` will call Etherfi `LiquidityPool.deposit`.

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L832-L835

```
function _ethTOeEth(uint256 _amount) internal returns (uint256) {
    // deposit returns exact amount of eETH
    return IeETHLiquidityPool(eETHLiquidityPool).deposit{value:
 ↪  _amount}(address(this));
}
```

The comment in `_ethTOeEth` states that the return value is the amount of eETH, but in reality Etherfi uses `mintShare` and returns the amount of shares.

https://github.com/etherfi-protocol/smart-contracts/blob/master/src/LiquidityPool.sol#L523-L533

```
function _deposit(address _recipient, uint256 _amountInLp, uint256
 ↪  _amountOutOfLp) internal returns (uint256) {
    totalValueInLp += uint128(_amountInLp);
    totalValueOutOfLp += uint128(_amountOutOfLp);
    uint256 amount = _amountInLp + _amountOutOfLp;
    uint256 share = _sharesForDepositAmount(amount);
    if (amount > type(uint128).max || amount == 0 || share == 0) revert
 ↪  InvalidAmount();

    eETH.mintShares(_recipient, share);

    return share;
}
```

`_depositPredefinedAsset` is called in `depositEth`, which in turn called `_eethTOweEth`, and the parameter is the share quantity of eETH returned by `_ethTOeEth`.

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L556-L557

```
} else if (_predefinedPool == PredefinedPool.weETH) {
    _finalAmount = _eethTOweEth(_amount);
```

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L843-L846

```
function _eethTOweEth(uint256 _amount) internal returns (uint256) {
    // wrap returns exact amount of weETH
    return IweETH(weETH).wrap(_amount);
}
```

However, in `weETH.wrap`, the parameter should be the actual amount of eETH rather than the amount of shares, as there is a conversion relationship between the actual amount and the amount of shares, they are not equal.

https://github.com/etherfi-protocol/smart-contracts/blob/master/src/WeETH.sol#L49-L55

```
function wrap(uint256 _eETHAmount) public returns (uint256) {
    require(_eETHAmount > 0, "weETH: cant wrap zero eETH");
    uint256 weEthAmount = liquidityPool.sharesForAmount(_eETHAmount);
    _mint(msg.sender, weEthAmount);
    eETH.transferFrom(msg.sender, address(this), _eETHAmount); //@audit amount,
↪   not share
    return weEthAmount;
}
```

`eETH.transferFrom` is to convert amount to share and then `transferShare`.

https://github.com/etherfi-protocol/smart-contracts/blob/master/src/EETH.sol#L111-L119 https://github.com/etherfi-protocol/smart-contracts/blob/master/src/EETH.sol#L143-L147

```
function _transfer(address _sender, address _recipient, uint256 _amount)
↪   internal {
    uint256 _sharesToTransfer = liquidityPool.sharesForAmount(_amount); //@audit
↪   convert amount to share
    _transferShares(_sender, _recipient, _sharesToTransfer);
    emit Transfer(_sender, _recipient, _amount);
}
```

As for why the current test cases pass, it is because `MockEETHLiquidityPool.deposit` uses `eEth.mint(msg.sender, mintAmount);`, which directly increases the amount of eETH and returns that amount directly, rather than returning the number of shares as in Etherfi.

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/mocks/MockeETHLiquidityPool.sol#L18-L26

```
function deposit(address _referral) external payable returns (uint256) {
    _referral;

    uint256 mintAmount = msg.value / 1001 * 1000;

    eEth.mint(msg.sender, mintAmount);
    return mintAmount;
}
```

Impact

As there is a conversion rate between the amount of eETH and the number of shares, which are not equal, the following situations may occur:

- If 100 ETH is deposited, 100 eETH and 90 eETH shares are obtained, then `weETH.wrap(90)` is executed, 10 eETH cannot be deposited into the pool, and the user loses assets.
- If 100 ETH is deposited, 100 eETH and 110 eETH shares are obtained, then `weETH.wrap(110)` is executed. Since there are only 100 eETH, the transaction will revert and the user will not be able to deposit assets.

Code Snippet

- https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L503-L516
- https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L832-L835
- https://github.com/etherfi-protocol/smart-contracts/blob/master/src/LiquidityPool.sol#L523-L533
- https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L556-L557
- https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L843-L846
- https://github.com/etherfi-protocol/smart-contracts/blob/master/src/WeETH.sol#L49-L55
- https://github.com/etherfi-protocol/smart-contracts/blob/master/src/EETH.sol#L111-L119
- https://github.com/etherfi-protocol/smart-contracts/blob/master/src/EETH.sol#L143-L147
- https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/mocks/MockeETHLiquidityPool.sol#L18-L26

Tool used

Manual Review

Recommendation

Like `_ethTOstEth`, return the difference of eETH balance instead of directly returning the result of `LiquidityPool.deposit`.

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L808-L813

```
function _ethTOstEth(uint256 _amount) internal returns (uint256) {
    // submit function does not return exact amount of stETH so we need to check
↪   balances
    uint256 balanceBefore = IERC20(stETH).balanceOf(address(this));
    IstETH(stETH).submit{value: _amount}(address(this));
    return (IERC20(stETH).balanceOf(address(this)) - balanceBefore);
}
```

Discussion

sherlock-admin4

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

0xmystery commented:

> valid because it's the shares that matter (best because report is most succint)

imp0wd3r

Escalate

As I mentioned in the report, this vulnerability can cause asset loss or deposit DoS, so it should be classified as high risk.

sherlock-admin3

> Escalate
>
> As I mentioned in the report, this vulnerability can cause asset loss or deposit DoS, so it should be classified as high risk.

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.

mystery0x

> Escalate
>
> As I mentioned in the report, this vulnerability can cause asset loss or deposit DoS, so it should be classified as high risk.

Agreed that this should be upgraded to `High`.

WangSecurity

Agree with the escalation, planning to accept and update to High severity.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/sophon-org/farming-contracts/commit/73adb6759b899e1a192fb5e28a5cd6202b5c3ff2

Evert0x

Result: High Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- imp0wd3r: accepted

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-2: Many cases `stEth::transferFrom` will transfer 1-2 less way, which would result in revert in consequent functions, because of not enough balance

Source: https://github.com/sherlock-audit/2024-05-sophon-judging/issues/63

The protocol has acknowledged this issue.

Found by

Bauchibred, EgisSecurity, zzykxx

Summary

When user calls `depositStEth`, he passes `_amount` param, which is set to `IERC20(stETH).safeTransferFrom()` func and then the sam `_amount` is passed down the chain:

```
IERC20(stETH).safeTransferFrom(
    msg.sender,
    address(this),
    _amount
);

_depositPredefinedAsset(_amount, _amount, _boostAmount, PredefinedPool.wstETH);
```

Vulnerability Detail

The probability of issue appearing is high and you can check in the following discussion. It has also been classified as a High severity on past contests: https://github.com/lidofinance/lido-dao/issues/442

`stETH` is using shares for tracking balances and it is a known issue that due to rounding error, transferred shares may be 1-2 wei less than `_amount` passed. This would revert on the following line as we have transferred `_amount - 1` and farming contract do not hold `stEth` funds:

```
function _stEthTOwstEth(uint256 _amount) internal returns (uint256) {
    // wrap returns exact amount of wstETH
    return IwstETH(wstETH).wrap(_amount);
}
```

The impact may be bigger if the staking contract is implemented by 3rd party protocol and expect this the function to be always fine.

Impact

- Contract functionality DoS

Code Snippet

https://github.com/sherlock-audit/2024-05-sophon/blob/05059e53755f24ae9e3a3bb2996de15df0289a6c/farming-contracts/contracts/farm/SophonFarming.sol#L474-L478

Tool used

Manual Review

Recommendation

Use lido recommendation to utilize `transferShares` function, so the `_amount` is realistic, or implement FoT approach, which compares the balance before and after the transfer.

Discussion

sherlock-admin4

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

0xmystery commented:

> valid because this parallels FOT as documented by Lido (best because the report is succinct and well documented)

mcbvictor0x

Fixed here: https://github.com/sophon-org/farming-contracts/commit/66e05ccb6cdfb8392b6a2bb4953931168cd247d3

Issue M-1: accPointsPerShare can reach a very large value leading to overflows

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

The protocol has acknowledged this issue.

Found by

ZdravkoHr.

Summary

Each pool tracks the amount of points that should be distributed for one share of its LP token in the accPointsPerShare variable. This variable can reach very large values causing integer overflows. This is dangerous as it puts the protocol's functionality at great risks.

Vulnerability Detail

This is the code that calculates accPointsPerShare. pointReward is a variable with 36 decimals precision because blockMultiplier and pointsPerBlock have both 18 decimals. The result is then divided by lpSupply.

```
uint256 blockMultiplier = _getBlockMultiplier(pool.lastRewardBlock,
↪   getBlockNumber());
uint256 pointReward =
    blockMultiplier *
    _pointsPerBlock *
    _allocPoint /
    totalAllocPoint;

pool.accPointsPerShare = pointReward /
    lpSupply +
    pool.accPointsPerShare;
```

The problem with this approach is that lpSupply can be a small value. It can happen either naturally (for example, tokens with low decimals, like USDC and USDT) or on purpose (by a malicious depositor).

The malicious depositor can deposit just 1 wei of the lp token and wait 1 block to update the accPointsPerShare variable. Since lpSupply will be equal to 1, accPointsPerShare will remain a value with 36 decimals.

Let's now have a look at pendingPoints()

```
function _pendingPoints(uint256 _pid, address _user) internal view returns
↪   (uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];

    uint256 accPointsPerShare = pool.accPointsPerShare * 1e18;

    uint256 lpSupply = pool.amount;
    if (getBlockNumber() > pool.lastRewardBlock && lpSupply != 0) {
        uint256 blockMultiplier = _getBlockMultiplier(pool.lastRewardBlock,
↪   getBlockNumber());

        uint256 pointReward =
            blockMultiplier *
            pointsPerBlock *
            pool.allocPoint /
            totalAllocPoint;

        accPointsPerShare = pointReward *
            1e18 /
            lpSupply +
            accPointsPerShare;
    }

    return user.amount *
        accPointsPerShare /
        1e36 +
        user.rewardSettled -
        user.rewardDebt;
}
```

`pool.accPointsPerShare` is multipled by 1e18 once again, resulting in 54 decimals. This is not the end, in the return statement this value will be multiplied once again by `user.amount`. Depending on the token's decimals, the value will be scaled again. For 18 decimals, the value will reach 72 decimals. This will result in unexpected overflows because `type(uint256).max < 1e78`

```
return user.amount *
    accPointsPerShare
```

Proof of Concept

In this test a first depositor deposits 1 wei worth of our Mock token to inflate the `accPointsPerShare`. In the next block, a honest depositor deposits `10_000` tokens. When `pendingPoints` is called, the transaction reverts because of an overflow.

```
function testOverflow() public {
    vm.startPrank(deployer);
    MockERC20 usdc = new MockERC20("Mock USDC Token", "MockUSDC", 18);
    usdc.mint(address(deployer), 10000e18);
    dai.mint(address(deployer), 1000e18);

    uint256 usdcId = sophonFarming.add(60000, address(usdc), "", false);

    dai.approve(address(sophonFarming), 1000e18);
    usdc.approve(address(sophonFarming), 10000e18);

    sophonFarming.depositDai(1000e18, 0);
    sophonFarming.deposit(usdcId, 1, 0);

    vm.roll(block.number + 1);

    sophonFarming.massUpdatePools();

    address[] memory users = new address[](1);
    users[0] = deployer;

    uint256[][] memory pendingPoints = sophonFarming.getPendingPoints(
        users
    );

    for (uint i = 0; i < pendingPoints.length; i++) {
        uint256 poolsLength = pendingPoints[i].length;

        for (uint j = 0; j < poolsLength; j++) {
            uint256 currentPoints = pendingPoints[i][j];
            console.log(currentPoints);
        }
    }

    sophonFarming.deposit(usdcId, 10000e18 - 1, 0);

    vm.roll(block.number + 1);

    sophonFarming.massUpdatePools();

    pendingPoints = sophonFarming.getPendingPoints(users);
    for (uint i = 0; i < pendingPoints.length; i++) {
        uint256 poolsLength = pendingPoints[i].length;

        for (uint j = 0; j < poolsLength; j++) {
            uint256 currentPoints = pendingPoints[i][j];
            console.log(currentPoints);
        }
    }
}
```

Impact

The `accPointsPerShare` variable becomes too large and breaks contract functionality

Code Snippet

https://github.com/sherlock-audit/2024-05-sophon/blob/05059e53755f24ae9e3a3bb2996de15df0289a6c/farming-contracts/contracts/farm/SophonFarming.sol#L423-L432 https://github.com/sherlock-audit/2024-05-sophon/blob/05059e53755f24ae9e3a3bb2996de15df0289a6c/farming-contracts/contracts/farm/SophonFarming.sol#L357C1-L385C1

Tool used

Manual Review

Recommendation

A possible solution may be to set a floor that a user has to deposit and also scale by a smaller value.

Discussion

sherlock-admin3

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

0xmystery commented:

> invalid because getPendingPoints() will not practically be used till point farming has ended. The higher precision adopted is by design

0xreadyplayer1 commented:

> even in the stated case - being the worst - the value goes uptil 1e72 and not 1e77 which is needed for overflow - the issue might exist but the watson was not able to clearly show the exploit about what happens if the price goes this large.

ZdravkoHr

Escalate The value 1e72 will be multiplied by the user amount, thus causing the overflow as in the provided proof of concept.

sherlock-admin3

> Escalate The value 1e72 will be multiplied by the user amount, thus causing the overflow as in the provided proof of concept.

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.

mystery0x

The extreme edge case isn't going to happen. You see, the user amount is only 1 wei when multiplying with the inflated accPointsPerShare (due to division by 1 wei of `lpSupply`).

So, when `user.amount` and `pool.amount` become sizable amounts (updated via the next _deposit(), `lpSupply` will no longer be 1 wei but rather whatever that has been last updated on pool.amount in `_pendingPoints()`.

WangSecurity

Do I understand correctly that the user still can deposit into the protocol by dividing their deposit into several ones? So they can successfully deposit all 10_000 tokens as the POC and later users will not have problems? @ZdravkoHr is it correct or not?

And @mystery0x as I understand, user.amount and pool.amount will be updated after the point when the revert happens, no? Cause if what you're saying is correct, the POC wouldn't work, no? Or the POC is wrong?

ZdravkoHr

When lpSupply becomes an 1e18 value, the accPointsPerShare will already be a very large value because in the updatePool function the amount is being added, not overwritten.

@WangSecurity, the problem is not that the user can't deposit, but that getPendingPoints will revert when the whole process depends on it

mystery0x

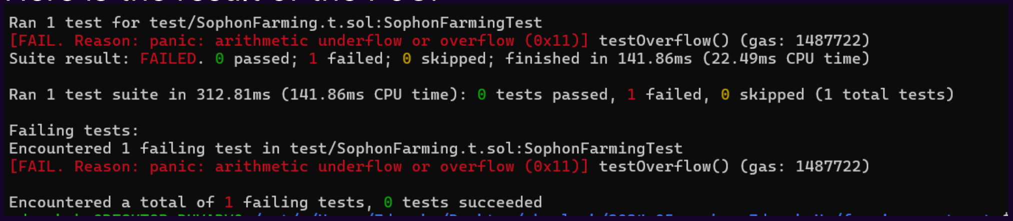
> When lpSupply becomes an 1e18 value, the accPointsPerShare will already be a very large value because in the updatePool function the amount is being added, not overwritten.

Can you run the test and show us the result reverting? I don't think it's going to be a problem since division by lpSupply (1e18) at this point is going to make accPointsPerShare diminished prior to having it multiplied to user.amount.

ZdravkoHr

The division by `1e18` will reduce the decimals of the `pointReward`, however this result will be added to the already big enough `pool.accPointsPerShare` value because there is an addition operation.

Here is the result of the PoC:



```
Ran 1 test for test/SophonFarming.t.sol:SophonFarmingTest
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)] testOverflow() (gas: 1487722)
Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 141.86ms (22.49ms CPU time)

Ran 1 test suite in 312.81ms (141.86ms CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/SophonFarming.t.sol:SophonFarmingTest
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)] testOverflow() (gas: 1487722)

Encountered a total of 1 failing tests, 0 tests succeeded
```

mystery0x

@ZdravkoHr Hmm... addition would cause overflow for the max value of uint256?

We agree that the first addend `user.amount * accPointsPerShare /1e36` is already curbed after the second call. How big can the second addend `user.rewardSettled` initially go? Can you plug in some practical numbers for these two addends and then the summation of both?

```
return user.amount *
    accPointsPerShare /
    1e36 +
    user.rewardSettled -
    user.rewardDebt;
```

ZdravkoHr

But how is the first calculation curbed if there is an addition in any subsequent call and not assignment? The value will babe large decimals initially and will continue growing, there seems to be no way to decrease it

mystery0x

> But how is the first calculation curbed if there is an addition in any subsequent call and not assignment? The value will babe large decimals initially and will continue growing, there seems to be no way to decrease it

`pointReward * 1e18 / lpSupply` is trivial as `lpSupply` is no longer 1 wei but 1e18:

```
accPointsPerShare = pointReward *
    1e18 /
    lpSupply +
    accPointsPerShare;
```

Just plug in some numbers, and I'm sure no issues are going to be found. You're not multiplying two big numbers (rather just adding) in the last arithmetic operation of function `_pendingPoints`:

```
return user.amount *
    accPointsPerShare /
    1e36 +
    user.rewardSettled -
    user.rewardDebt;
```

ZdravkoHr

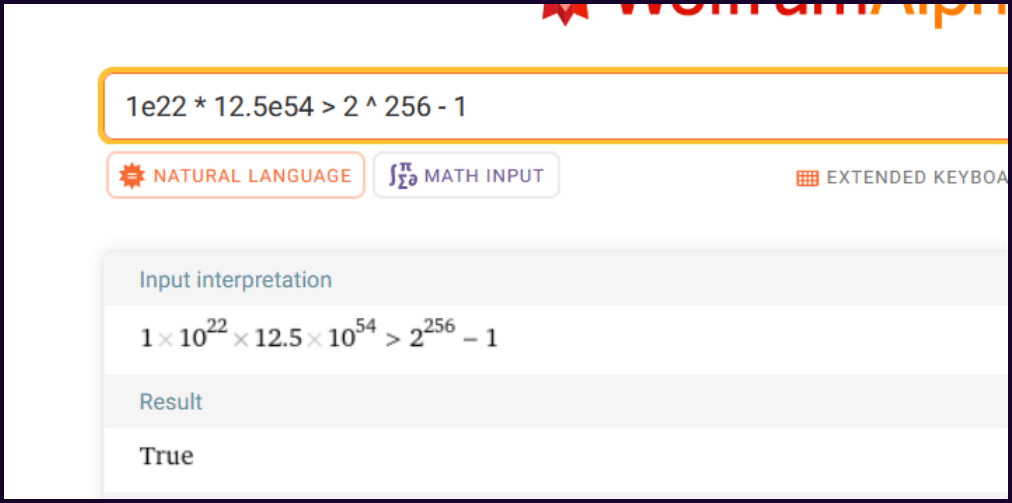
With the default values from the tests, pointsPerBlock = 25e18 and allocation of 50%. accPointsPerShare = 12.5e36

Users deposit and this value becomes > 12.5e36

Then we go here

```
uint256 accPointsPerShare = pool.accPointsPerShare * 1e18;
 ...
return user.amount *
    accPointsPerShare /
    1e36 +
    user.rewardSettled -
    user.rewardDebt;
```

Let user amount be 10_000e18 = 1e22. Then we have `1e22 * 12.5e54` which is greater than `type(uint256.max)`


1e22 * 12.5e54 > 2 ^ 256 - 1

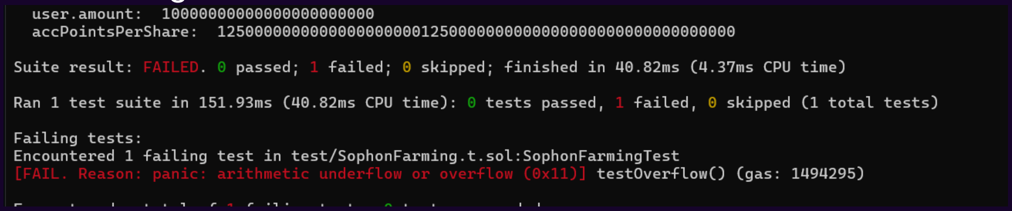
Input interpretation

$1 \times 10^{22} \times 12.5 \times 10^{54} > 2^{256} - 1$

Result

True

Here are logs from the PoC:


```
  user.amount:  10000000000000000000000
  accPointsPerShare:  1250000000000000000000012500000000000000000000000000000000

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 40.82ms (4.37ms CPU time)

Ran 1 test suite in 151.93ms (40.82ms CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/SophonFarming.t.sol:SophonFarmingTest
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)] testOverflow() (gas: 1494295)
```

mystery0x

Ah, ok. @RomanHiden you might want to look into this edge issue in `_pendingPoints()`.

WangSecurity

Based on the above discussion, I agree with the escalation.

Planning to accept it and validate with medium severity. Are there any duplicates of it?

ZdravkoHr

@WangSecurity, just searched for `getPendingPoints` and couldn't find a duplicate (there were 16 results). Maybe if @mystery0x knows of some

mystery0x

Many other reports touched on the topic but none of them identified the 1 wei culprit from the first and only depositor in the initial block(s).

It's noteworthy that the suggested mitigation, "to set a floor that a user has to deposit" will also solve the issues associated with #85 and #177.

WangSecurity

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ZdravkoHr: accepted

mcbvictor0x

> The protocol team fixed this issue in the following PRs/commits: sophon-org/farming-contracts@f0b82fd

Can this issue be closed?

Issue M-2: Protocol won't be eligible for referral rewards for depositing ETH

Source: https://github.com/sherlock-audit/2024-05-sophon-judging/issues/92

Found by

h2134

Summary

Protocol won't be eligible for referral rewards for depositing ETH, as the referral account is not properly set.

Vulnerability Detail

When user sends ETH to contract **SophonFarming**, or calls ETH deposit functions, protocol will first deposit ETH on external protocols (Lido or Ether.fi) for minting staking shares, then warp the shares to the predefined assets (wstETH or weETH), then user starts to earn reward points.

The external protocols allows caller to pass referral argument when depositing ETH, and referral account can be eligible for referral rewards if it is valid. Let's take Ether.fi for example:

```
function _ethTOeEth(uint256 _amount) internal returns (uint256) {
    // deposit returns exact amount of eETH
    return IeETHLiquidityPool(eETHLiquidityPool).deposit{value:
↪   _amount}(address(this));
}
```

To convert ETH to eETH, the function `deposit` in IeETHLiquidityPool is called. In fact, **IeETHLiquidityPool** exposes 2 deposit functions for users to deposit ETH, which have different signatures:

```
// Used by eETH staking flow
function deposit() external payable returns (uint256) {
    return deposit(address(0));
}

// Used by eETH staking flow
function deposit(address _referral) public payable whenNotPaused returns
↪   (uint256) {
    require(_isWhitelisted(msg.sender), "Invalid User");
    emit Deposit(msg.sender, msg.value, SourceOfFunds.EETH, _referral);

    return _deposit(msg.sender, msg.value, 0);
}
```

The `_referral` parameter in the second deposit function indicates the referral account which will be eligible for referral rewards, as stated by ether.fi here:

> This referral program covers both ether.fi and ether.fan, each 0.1 ETH staked via ether.fi or ether.fan earns the person who stakes > and the the person who referred 100 loyalty points.
>
> Note: Referral points may take up to 2 hours to display in your Portfolio.

Apparently, by calling the second deposit function and passing **address(this)** as `_referral` argument, our protocol expects to receive the referral rewards, however, this makes the referral as the same account as the depositor itself (msg.sender), this is invalid to ether.fi and no rewards will be granted to the account which uses one's own referral code for depositing.

Similarly, protocol won't receive referral rewards from Lido as it set referral to itself when submit to deposit ETH:

```
    function _ethTOstEth(uint256 _amount) internal returns (uint256) {
        // submit function does not return exact amount of stETH so we need to
↪   check balances
        uint256 balanceBefore = IERC20(stETH).balanceOf(address(this));
@=>     IstETH(stETH).submit{value: _amount}(address(this));
        return (IERC20(stETH).balanceOf(address(this)) - balanceBefore);
    }
```

Impact

Protocol won't be eligible for referral rewards as expected, this can be significant value leak to the protocol.

Code Snippet

https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L811 https://github.com/sherlock-audit/2024-05-sophon/blob/main/farming-contracts/contracts/farm/SophonFarming.sol#L834

Tool used

Manual Review

Recommendation

User `owner` account as referral instead of the caller contract itself.

```
    function _ethTOstEth(uint256 _amount) internal returns (uint256) {
        // submit function does not return exact amount of stETH so we need to
↪   check balances
        uint256 balanceBefore = IERC20(stETH).balanceOf(address(this));
-       IstETH(stETH).submit{value: _amount}(address(this));
+       IstETH(stETH).submit{value: _amount}(owner());
        return (IERC20(stETH).balanceOf(address(this)) - balanceBefore);
    }
```

```
    function _ethTOeEth(uint256 _amount) internal returns (uint256) {
        // deposit returns exact amount of eETH
-       return IeETHLiquidityPool(eETHLiquidityPool).deposit{value:
↪   _amount}(address(this));
+       return IeETHLiquidityPool(eETHLiquidityPool).deposit{value:
↪   _amount}(owner());
    }
```

Discussion

sherlock-admin3

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

0xmystery commented:

> valid because referral should indeed be an EOA which can be multisig

ZdravkoHr

Escalate This issue looks like Oportunity Loss which is not accepted as H/M by Sherlock. For example, a contract not opting-in for gas yield on Blast would not be a valid H/M.

sherlock-admin3

> Escalate This issue looks like Oportunity Loss which is not accepted as H/M by Sherlock. For example, a contract not opting-in for gas yield on Blast would not be a valid H/M.

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.

mystery0x

> Escalate This issue looks like Oportunity Loss which is not accepted as H/M by Sherlock. For example, a contract not opting-in for gas yield on Blast would not be a valid H/M.

I think this will mean a lot more than just `Opportunity Loss` given that many transactions are going to route through these calls.

WangSecurity

@mystery0x could you please iterate why you think it's not an Opportunity loss?

0xh2134

This issue is not opportunity loss because:

1. It's not caused by loss of functionality but the wrong argument in the code;
2. It's protocol design to receive referral rewards, this can be seen from the code implementation (and sponsor's confirmation of this issue)

This issue may fall into this category:

> Loss of airdrops or liquidity fees or any other rewards that are not part of the original protocol design is not considered a valid high/medium

However because referral rewards is the protocol design, so this issue is valid.

mystery0x

@WangSecurity @0xh2134 addresses a valid point because of the erring integration (In the example provided by Sherlock Criteria for Issue Validity doc, users do not have an option to claim their airdrops whereas the referral bonus in this context is readily claimable albeit is denied due to wrong input of function argument.)

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/sophon-org/farming-contracts/commit/73adb6759b899e1a192fb5e28a5cd6202b5c3ff2

WangSecurity

I agree that it's not an opportunity loss and doesn't qualify for a loss of rewards, since the protocol's code wants to receive them, but it doesn't due to code issue. Hence, I believe it's part of the original protocol design.

Planning to reject the escalation and leave the issue as it is.

ZdravkoHr

Isn't the main reward from stETH coming from beacon chain staking rewards and thia refferal fee is just an additional bonus?

WangSecurity

As I understand yes, but as I see in the code, these rewards are part of the protocol's design and they want to accrue them, but it won't happen due to an issue in the code. Or am I missing something?

If not, decision remains the same, reject the escalation and leave the issue as it is.

ZdravkoHr

How do we understand that they want to accrue them?

WangSecurity

By the fact that they submit an address to receive the rewards, instead of address(0) for example, it would be fair if they didn't care. But, actually, looking at this issue a couple of times more, I'm not sure about it. Is it possible to see the code implementation of that submit function? I think it might shed some light on the problem.

ZdravkoHr

I found this function from the LIDO contract it seems the `referral` is used only in event emission.

```
function _submit(address _referral) internal returns (uint256) {
    require(msg.value != 0, "ZERO_DEPOSIT");

    StakeLimitState.Data memory stakeLimitData =
↪   STAKING_STATE_POSITION.getStorageStakeLimitStruct();
    // There is an invariant that protocol pause also implies staking pause.
    // Thus, no need to check protocol pause explicitly.
    require(!stakeLimitData.isStakingPaused(), "STAKING_PAUSED");

    if (stakeLimitData.isStakingLimitSet()) {
        uint256 currentStakeLimit = stakeLimitData.calculateCurrentStakeLimit();

        require(msg.value <= currentStakeLimit, "STAKE_LIMIT");

        STAKING_STATE_POSITION.setStorageStakeLimitStruct(stakeLimitData.updateP ⌋
↪   revStakeLimit(currentStakeLimit - msg.value));
    }

    uint256 sharesAmount = getSharesByPooledEth(msg.value);

    _mintShares(msg.sender, sharesAmount);
    _setBufferedEther(_getBufferedEther().add(msg.value));
    emit Submitted(msg.sender, msg.value, _referral);

    _emitTransferAfterMintingShares(msg.sender, sharesAmount);
    return sharesAmount;
}
```

Also, in this article it's explained that the rewards go to the specified address and not to the calling one - I didn't know that :D

> To participate, users simply need to generate a referral URL from their Ethereum addresses and share the link with others. The referrer is rewarded with payback in LDO, Lido's native token, based on the amount of ETH staked by users who join through the referral link.

0xh2134

The way I see it, it's obvious that they want to accrue referral rewards:

1. When deposit into Lido, call `submit(address _referral)` function with `_referral` being `address(this)` instead of `address(0)`
2. When deposit into Ether.fi, call `deposit(address _referral)` function with `_referral` being `address(this)`, instead of calling the default `deposit()` function

The deposit events are processed by Lido / Ether.fi offline, and referral rewards are distributed to `_referral`. However, in the context of deposit event, the `msg.sender` is **SophonFarming** contract and `_referral` is also **SophonFarming** contract, as I confirmed with Ether.fi, no referral rewards will be distributed if `_referral` is the same as the `msg.sender`.

So, because they want to accrue referral rewards, they need to set `_referral` to a different address other than `msg.sender`, for example, the owner, then the owner can send referral rewards back to **SophonFarming** contract.

BTW, I believe it's highly likely that the referral rewards would be used as rewards to point holders, that's why the want to accrue them.

WangSecurity

Thanks for both of your answers. I also confirmed with the sponsor that these referral rewards are indeed part of their original design and they intended to receive them. I know that some may say it was known only after the contest, but I believe in this situation it's quite unclear from the code if they intended to receive them and all the watsons could ask sponsors about it during the contest. With that said, since the team intends to receive these rewards, but made a code issue leading to losing these rewards, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- ZdravkoHr: rejected

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-3: `setStartBlock()` doesn't change the block at which already existing pools will start accumulating points

Source: https://github.com/sherlock-audit/2024-05-sophon-judging/issues/108

The protocol has acknowledged this issue.

Found by

0xAadi, EgisSecurity, KupiaSec, MightyRaju, ZdravkoHr., araj, blackhole, dhank, h2134, jecikpo, serial-coder, underdog, utsav, whitehair0330, yamato, zzykxx

Summary

The function setStartBlock() can be called by the owner to change the block number at which points will start accumulating. When it's called, the block at which already existing pools will start accumulating points will not change. Already existing pools will:

1. Start accumulating points from the old `startBlock` if the new `startBlock` is set after the old one.
2. Not accumulate rewards until the old `startBlock` is reached if the new `startBlock` is set before the old one.

Vulnerability Detail

This happens because updatePool() considers the pool `lastRewardBlock` as the block number from which points should start accumulating and setStartBlock() never updates the `lastRewardBlock` of the already existing pools to the new `startBlock`.

POC

Runnable POC that showcases point `1` explained above. Can be copy-pasted in `SophonFarming.t.sol`:

```
function test_SettingStartBlockDoesntUpdatePools() public {
    address alice = makeAddr("alice");
    uint256 amountToDeposit = sDAI.convertToAssets(1e18);

    vm.prank(alice);
    dai.approve(address(sophonFarming), type(uint256).max);
    deal(address(dai), alice, amountToDeposit);
    //-> Pools original `startBlock` is `1`
    //-> Admin changes `startBlock` to `100`
    vm.prank(deployer);
    sophonFarming.setStartBlock(100);

    //-> Alice deposits at block `90`, which is after the previous `startBlock`
 ↪  (1) but before the current `startBlock` (100)
    vm.roll(90);
    vm.prank(alice);
    sophonFarming.depositDai(amountToDeposit, 0);

    //-> After 9 blocks, at block `99`, Alice has accumulated rewards but she
 ↪  shouldn't have because the current `startBlock` (100) has not been reached
 ↪  yet
    vm.roll(99);
    vm.prank(alice);
    sophonFarming.withdraw(0, type(uint256).max);
    assertEq(sophonFarming.pendingPoints(0, alice), 749999999999999999);
}
```

Can be run with:

```
forge test --match-test test_SettingStartBlockDoesntUpdatePools -vvvvv
```

Impact

When setStartBlock() is called the block at which already existing pools will start accumulating points will not change.

Code Snippet

Tool used

Manual Review

Recommendation

In setStartBlock() loop over all of the existing pools and adjust each pool `lastRewardBlock` to the new `startBlock`. Furthermore setStartBlock() should revert if the new `startBlock` is lower than the current `block.number` as this would create problems in points distribution accounting if the above fix is implemented.

Discussion

sherlock-admin4

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

0xmystery commented:

> valid because lastRewardBlock for each pool should indeed sync with the latest startBlock (best because the report is succinct and comprehensive explaining each of the two scenarios)

mcbvictor0x

Issue is no longer relevant since we removed the global startBlock setting: https://github.com/sophon-org/farming-contracts/commit/18b59d5d1a987caf821245ab83505ca6f62904e6

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.