SHERLOCK

Security Review For Spectra Finance

Collaborative Audit Prepared For: **Spectra Finance**
Lead Security Expert(s): **hildingr**
panprog

Date Audited: **September 21 - September 26, 2025**
Final Commit: **d2a62da**

Introduction

Spectra is an EVM-centric protocol for interest rate derivatives with an easy-to-use flagship app. The Spectra protocol is permissionless, meaning its services are entirely open for public use. Anyone can create new markets at will, swap yield derivatives, or become a liquidity provider. Spectra's Yield Token and Principal Token, minted on top of ERC-4626 interest-bearing tokens, are core protocol primitives. Spectra separates the right to principal from future yield via a process called yield tokenization. This process unlocks new financial possibilities beyond standalone interest-bearing tokens.

Scope

Repository: perspectivefi/metavaults-V1

Audited Commit: e8f94ca9f0bb60a8c44f1e08e52983960141222c

Final Commit: d2a62da161819d5666796030cc604daf7b73e29f

Files:

- src/gatekeepers/PoolGatekeeper.sol
- src/gatekeepers/PrincipalTokenGatekeeper.sol
- src/MetavaultsRegistry.sol
- src/MetaVaultWrapper.sol
- src/zaps/CurveLiquidityZap.sol
- src/zaps/PTZap.sol

Final Commit Hash

d2a62da161819d5666796030cc604daf7b73e29f

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	2	9

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: `totalVirtualSupply` and `totalVirtualInfraVaultSupply` can easily fail to account wrapper claimed deposits or redeems, bricking the wrapper with all user funds locked in it

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/9

Summary

The `WrapperState` includes virtual supply amounts:

```
struct WrapperState {
    // Total virtual supply for rate calculations
    uint256 totalVirtualSupply;
    // Total virtual infra vault supply for rate calculations
    uint256 totalVirtualInfraVaultSupply;
    // Address of the underlying infrastructure vault
    address infraVault;
    // Address of the underlying asset token
    address underlying;
    // Mapping to track operator permissions for controllers
    mapping(address controller => mapping(address operator => bool)) isOperator;
}
```

The only code which updates the virtual supply is when wrapper claims deposit or redeem from the Amphor vault:

```
            // Claim deposit from Amphor
            uint256 shares = IAmphor(W.infraVault).claimDeposit(address(this));
            // Update virtual infra vault supply
            W.totalVirtualInfraVaultSupply += shares;
            // Update virtual total supply
            W.totalVirtualSupply += shares;
...
            // Claim redeem from Amphor
            uint256 assets = IAmphor(W.infraVault).claimRedeem(address(this));
            // Decrease virtual infra vault supply
            W.totalVirtualInfraVaultSupply -= claimableRedeem;
            // Decrease virtual total supply
            W.totalVirtualSupply -= claimableRedeem;
```

The issue is that there are multiple cases when wrapper's deposit or redeem requests in Amphor are claimed without wrapper ever executing the code to update virtual supply:

- Anyone can deposit or redeem tiny amount on behalf of wrapper directly in

Amphor vault (setting `receiver = wrapper`). This operation will first claim deposit (redeem) requests of the wrapper, thus wrapper's amount to be claimed will be 0 and it will fail to update virtual supply.

- Anyone can deposit or redeem any amount on behalf of wrapper directly in Amphor vault. This will increase shares or assets in wrapper's deposit (redeem) request which is not accounted for in the virtual supply.
- Wrapper only claims its deposit (redeem) (and updates virtual supply) if the user who is depositing (redeeming) has pending deposit (redeem) request. If this is a new user, virtual supply is not updated, but the wrapper's pending deposit (redeem) is auto-claimed anyway.
- When wrapper claims deposit, it verifies that `infraVault.claimableDepositBalanceInAsset(wrapper) > 0` to claim, however this function converts amounts twice (deposit assets => shares => assets), rounding down in each step. This means that when `shares > 0`, it's possible that `assets == 0` and deposit won't be claimed by the wrapper. It will later be auto-claimed at the time the new deposit request is created, failing to update virtual supply:

```
function _claimPendingAmphorDeposit() internal {
    WrapperState storage W = _getWrapperStorage();
    // Preview claimable deposit amount
    uint256 claimableDeposit = IAmphor(W.infraVault).claimableDepositBalanceInAsset(
        address(this)
    );
    // Process claim if amount is greater than 0
    if (claimableDeposit > 0) {
        // Claim deposit from Amphor
        uint256 shares = IAmphor(W.infraVault).claimDeposit(address(this));
```

Root Cause

The only code where virtual supply is updated is when claiming wrapper deposit or redeem request:
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L774-L779
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L831-L836

Another issue is the usage of `claimableDepositBalanceInAsset` function, which double-converts the assets => shares => assets, rounding it down:
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L769

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

Example 1: via claiming deposit directly in Amphor vault

- Request deposit into wrapper or wait from the other users to do it
- Immediately after the Amphor vault is settled and epoch advances, directly deposit 1 wei of asset into Amphor vault, setting `receiver = wrapper`
- This auto-claims the deposit request on behalf of wrapper, skipping the virtual supply update
- Now the virtual supply is significantly smaller than actual wrapper shares
- If the difference of virtual supply and actual shares is not large enough, repeat the same process until the difference is large enough
- Request redeem from wrapper, and after Amphor vault is settled, any request to redeem or redeem attempt will revert due to underflow when trying to decrease virtual supply (as the redeem amount will be larger than virtual supply due to de-sync)

Example 2: via doing redeem directly in Amphor vault

- Request large redeem request directly in Amphor vault, setting `receiver = wrapper`
- After the Amphor vault is settled, all attempts to request redeem or redeem in wrapper will revert with underflow, because the claimed amounts will be greater than virtual supply

Example 3: via different users

- User1 requests deposit to wrapper
- Amphor vault is settled
- User2 requests deposit to wrapper (this auto-claims wrapper deposit without updating virtual supply)
- User1 claims deposit and requests to redeem
- Amphor vault is settled
- User1 tries to redeem, but it reverts trying to reduce virtual supply

Example 4: via incorrect function to determine if deposit should be claimed

- Happens only when `0.5 < share price < 1`
- Request to deposit tiny asset amount via wrapper (e.g. 1 wei)
- Wait for the Amphor vault to settle

- Claim deposit (1 share, which is worth 0 assets)
- Due to incorrect function usage, the wrapper won't update virtual supply, which will desync from the Amphor vault amounts by a tiny amount.

Impact

- The wrapper is temporarily bricked with users unable to redeem their shares
- The virtual supply accounting is totally messed and disconnected from any real values, basically being unusable for any purpose at it can be any random amount

Note: the wrapper will only be bricked temporarily, until deposits increase virtual supply or direct tiny redeem in the Amphor vault is done to claim redeem on behalf of the wrapper, which will unlock redeems again. Still, it can stop normal vault operation and can cause panic to users who will be unable to claim or initiate their redeems.

Proof of concept

This is Example 1 done in the test. Add this test to `Redeem.t.sol`:

```
// - request deposit
// after it can be claimed:
// - request deposit 1 wei directly in the vault with receiver set to wrapper, this
↪  will claim deposit without updating virtual supply
// - request redeem
// after it can be claimed
// - wrapper is bricked. Any function will try to redeem first, but it will
↪  underflow trying to reduce virtual supply which didn't increase first
function test_redeemBrickWrapper1() public {
    uint256 testAmount = 5e21;

    dealAndApprove(MOCK_USER_1, UNDERLYING, testAmount + 1);

    vm.prank(MOCK_USER_1);
    IERC7540(wrapper).requestDeposit(testAmount, MOCK_USER_1, MOCK_USER_1);

    _updateInfrastructureVault();

    vm.startPrank(MOCK_USER_1);
    IERC20(UNDERLYING).approve(amphorVault, testAmount);
    IAmphor(amphorVault).requestDeposit(1, wrapper, MOCK_USER_1, "");

    // claim deposit
    IERC7540(wrapper).deposit(AMOUNT, MOCK_USER_1, MOCK_USER_1);
    vm.stopPrank();

    // Request redeem
    uint256 userShares = IERC7540(wrapper).balanceOf(MOCK_USER_1);
```

```
    vm.prank(MOCK_USER_1);
    IERC7540(wrapper).requestRedeem(userShares, MOCK_USER_1, MOCK_USER_1);
    _updateInfrastructureVault();

    // wrapper is now bricked, nobody can redeem
    vm.prank(MOCK_USER_1);
    IERC7540(wrapper).redeem(0, MOCK_USER_1, MOCK_USER_1);
}
```

The test will revert with underflow when trying to reduce virtual supply.

Mitigation

It appears to be impossible to accurately track all deposits and redeems by the wrapper since direct interactions with the Amphor vaults are possible. The recommendation is to get rid of virtual supply calculations altogether. With the current Amphor vault it appears to be useless anyway: both virtual supply amounts are always updated at the same time by the same amount, so `_previewWrap` and `_previewUnwrap` always return the same amount (multiplying by `1.0`).

These virtual supply amounts might be useful after migration, but not before it. For this reason, it makes sense to simply calculate these values off-chain and set them during the migration process, rather than trying to calculate it on the fly during normal operations.

Discussion

petarcalic99

Reviewed and we agree that this is an issue blocking temporarily users as its fixable using a redeem on behalf of the wrapper. However even tho we acknowledge the unexpected behaviour of the protocol we believe this is more of a medium severity as there is no risk of loss of funds for the users/ permanent locking. Let us know your opinion on this.

panprog

Ok, agree. There is no loss of funds, only temporary block and malfunctioning functionality. Changed severity to Medium.

Issue M-2: `MetaVaultWrapper.redeem` sends the funds to `controller` instead of `receiver`

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/11

Summary

The `MetaVaultWrapper.redeem(shares, receiver, controller)` function is expected to claim `controller`'s redeem request and send the assets received to `receiver`. However, the `receiver` is only included in `emit`, but the actual funds are sent to `controller`:

```
// Claim pending redeem and get results
(uint256 claimableShares, uint256 assets) = _claimPendingRedeem(controller,
↪  controller);

emit IERC4626.Withdraw(address(this), receiver, controller, assets,
↪  claimableShares);
```

Root Cause

Using `controller` both as `owner` and `receiver`: https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L595

Internal Pre-conditions

`MetaVaultWrapper.redeem` called with `receiver` different from `controller`.

External Pre-conditions

None.

Attack Path

Happens always by itself.

Impact

`receiver` unexpectedly not receiving the assets it should.

Mitigation

Claim redeem to receiver:

```
(uint256 claimableShares, uint256 assets) = _claimPendingRedeem(controller,
↪  receiver);
```

Discussion

petarcalic99

Agreed, it should be reciever who recieves the assets. Will be fixed.

Issue L-1: `MetaVaultWrapper` is not compliant with the `ERC-7540` standard due to lack of `supportsInterface` function

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/7

Summary

The ERC7540 standard states:

> Smart contracts implementing this Vault standard MUST implement the ERC-165 supportsInterface function.
> All asynchronous Vaults MUST return the constant value true if either 0xe3bc4e65 (representing the operator methods that all ERC-7540 Vaults implement) or 0x2f0a18c5 (representing the ERC-7575 interface) is passed through the interfaceID argument.
> Asynchronous deposit Vaults MUST return the constant value true if 0xce3bbe50 is passed through the interfaceID argument.
> Asynchronous redemption Vaults MUST return the constant value true if 0x620ee8e4 is passed through the interfaceID argument.

However, the `MetaVaultWrapper` doesn't have `supportsInterface` function, thus it's not compliant with `ERC-7540` standard.

Root Cause

Lack of `interfaceSupports` function.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

External contract queries if `MetaVaultWrapper` supports `ERC-7540`, but due to revert of this function, incorrectly treats it as if it doesn't support `ERC-7540`.

Impact

Lack of compliance of the `ERC-7540` standard. Some external contracts / integrations failing due to `interfaceSupport` checks.

Proof of concept

Add this test to `7540Properties.t.sol`:

```
...
interface IERC165 {
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
...
    function test_supportsInterface() view external {
        assertEq(IERC165(wrapper).supportsInterface(0xe3bc4e65), true,
        ↪  "supportsInterface (operator)");
        assertEq(IERC165(wrapper).supportsInterface(0x2f0a18c5), true,
        ↪  "supportsInterface (ERC7575)");
        assertEq(IERC165(wrapper).supportsInterface(0xce3bbe50), true,
        ↪  "supportsInterface (async deposit)");
        assertEq(IERC165(wrapper).supportsInterface(0x620ee8e4), true,
        ↪  "supportsInterface (async withdrawal)");
    }
```

Mitigation

Add `supportsInterface` function to support all the required `interfaceIDs`.

Discussion

petarcalic99

Agreed, we will add this the support interface function

Issue L-2: `MetaVaultWrapper` is not compliant with the `ERC-7540` standard due to lack of `share` method

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/8

Summary

The ERC7540 standard states:

> Smart contracts implementing this Vault standard MUST implement the ERC-7575 standard (in particular the share method).

However, the `MetaVaultWrapper` doesn't have `share` method / public property, thus it's not compliant with `ERC-7540` standard. In `MetaVaultWrapper` case, this function should return the `MetaVaultWrapper`'s own address.

Root Cause

Lack of `share` method.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

External contract tries to fetch the share token from `MetaVaultWrapper`, but reverts.

Impact

Lack of compliance of the `ERC-7540` standard. Some external contracts / integrations unexpectedly reverting.

Proof of concept

Add this test to `7540Properties.t.sol`:

```
...
interface IERC7575 {
    function share() external view returns (address);
}
...
    function test_supports7575() view external {
        address _share = IERC7575(wrapper).share();
        assertNotEq(_share, address(0), "supports share() method");
    }
```

Mitigation

Add `share` public function to return `address(this)`.

Discussion

petarcalic99

Agreed, we will add it.

petarcalic99

Actually the share function belongs to the 7575 interface which we don't implement. There was a typo in the name of a test folder which maybe caused the confusion

panprog

> Actually the share function belongs to the 7575 interface which we don't implement.

Refer to ERC-7540 requirements

> Smart contracts implementing this Vault standard MUST implement the ERC-7575 standard (in particular the share method).

If you intend to comply with ERC-7540, you **MUST** implement ERC-7575 and `share` method in particular.

It's not like a lot is required from you. The only requirements to implement 7575:

- add `share` (just a public storage variable holding contract's own address)
- `supportsInterface` returning `true` for `0xf815c03d`

That's all you need to comply here.

petarcalic99

Yes just saw it in the 7540 specs. Will add it.

Issue L-3: Incorrect comment in `MetaVaultWrapper.initialize`

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/10

Summary

The following comment before the `MetaVaultWrapper.initialize` function is incorrect:

```
/// @param _underlying      Underlying ERC20; if zero, the function will try
 ↪  infraVault_.asset()
```

The function simply reverts if `_underlying == 0`:

```
if (_infraVault == address(0) || _underlying == address(0)) {
    revert IERC7540.ZeroAddress();
}
```

Root Cause

Incorrect comment:
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L111

Discussion

petarcalic99

Agreed, will be updated

Issue L-4: `MetaVaultWrapper.decreaseDepositRequest` and `MetaVaultWrapper.decreaseRedeemRequest` do not revert when trying to reduce non-existing request

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/12

Summary

When deposit or redeem request is decreased in `MetaVaultWrapper`, any user-supplied amount will return early if existing request has 0 assets (shares):

```
// Return early if no pending balance
if (pending == 0) return;

// Validate decrease amount is not zero
if (assets == 0) {
    revert ZeroDecreaseAmount();
}

// Validate decrease amount doesn't exceed pending
if (assets > pending) {
    revert DecreaseAmountExceedsPending(pending, assets);
}
```

The correct behaviour should be to revert since it is not possible to reduce non-existing request.

Root Cause

Early return when pending request amount is 0:
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L409
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L541

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

User tries to reduce non-existing deposit or redeem request.

Impact

Nothing is done silently while it should revert.

Mitigation

Remove the check for `pending == 0` condition as it doesn't provide anything valuable while preventing correct revert.

Issue L-5: `MetaVaultWrapper.maxDeposit` and `MetaVaultWrapper.maxRedeem` return incorrect value when latest user request was in the current epoch

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/13

Summary

The functions `maxDeposit` and `maxRedeem` return the max claim deposit/redeem amount for the user. Since user can't claim deposit/redeem requests made in the current epoch, it should return 0 for the current epoch, but it will return the current request amount instead:

```
function maxDeposit(
    address receiver
) public view override(ERC4626Upgradeable, IERC4626) returns (uint256 maxAssets) {
    // Return 0 if contract is paused
    if (paused()) {
        return 0;
    }

    // Get user's last deposit request epoch
    uint256 userEpoch = _getAmphorStorage().lastDepositRequestId[receiver];
    // Return claimable deposit balance for that epoch
    return _getAmphorStorage().epochs[userEpoch].depositRequestBalance[receiver];
}
```

Root Cause

No check if user epoch is the current epoch:
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L239-L242
https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L253-L256

Internal Pre-conditions

User has deposited or redeemed in the current epoch.

External Pre-conditions

None.

Attack Path

- `maxDeposit` or `maxRedeem` returns current request amount instead of 0.
- `deposit` or `redeem` with non-0 amount will pass the `maxDeposit` (`maxRedeem`) check, but will still revert due to user epoch being current epoch.

Impact

The only impact right now is the incorrect revert reason for the `deposit` or `redeem`: instead of reverting with the `ERC7540ExceededMaxDeposit` or `ERC7540ExceededMaxRedeem`, it reverts with the `NoClaimAvailable`.

Proof Of Concept

Add this test to `Deposit.t.sol`:

```
function test_maxDeposit() public {
    uint256 userBalance = IERC20(UNDERLYING).balanceOf(MOCK_USER_1);
    vm.prank(MOCK_USER_1);
    IERC7540(wrapper).requestDeposit(userBalance, MOCK_USER_1, MOCK_USER_1);

    _updateInfrastructureVault();

    vm.prank(MOCK_USER_1);
    IERC7540(wrapper).deposit(userBalance, MOCK_USER_1, MOCK_USER_1);
}
```

Mitigation

Return 0 if user epoch is current epoch.

Discussion

petarcalic99

Agreed, we will fix this

Issue L-6: Incorrect update of epochId in _claimPendingRedeem()

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/14

Summary

In `_claimPendingRedeem()` the `epochId` never updated to new id as it is in `_claimPendingDeposit()`. The id is instead set to the old idea again.

Vulnerability Detail

We see here

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L818

The the ID is not updated as it is here

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L757

When `_claimPendingRedeem()` is called from `redeem()` the lastDepositRequestId will remain with the old ID.

Impact

Currently no impact as the balance is set to 0.

Code Snippet

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L818

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L757

Tool Used

Manual Review

Recommendation

Update to the new epochId

Issue L-7: Redundant check in deposit and redeem

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/15

Summary

All assets/shares are always deposited or redeemed the two following checks against the `asset` parameter are therefore redundant

Vulnerability Detail

All assets/shares are always deposited or redeemed the two following checks are therefore redundant

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L454-L456

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L588-L590

`WrapperMetaVault` does not supported partial or user entered assets/shares when depositing and redeeming and will always use the total amount.

Impact

Unnecessary check as the assets/shares parameter have no impact.

Code Snippet

Tool Used

Manual Review

Recommendation

Remove check

Discussion

petarcalic99

Agreed, we will remove the redundant check

Issue L-8: Curator could potential steal from the vault with unbalanced liquidity deposits

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/16

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

In the `CurveLiquidityZap.sol` the curator can set arbitrary values for `tokens`, `ibts`, `minYTs` and `minLPs` as such it is possible to add liquidity to curve with a unbalanced amount. Vault would receive less LP due to slippage and increased fee and the curator could profit by arbitraging the pool directly after.

Vulnerability Detail

The curator sets `tokens`, `ibts`, `minYTs` and `minLPs` . With these parameters curator decides the ratio of `ibt` and `pt` tokens to deposit into the curve pool

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/zaps/CurveLiquidityZap.sol#L127

By depositing an unbalanced amount they will receive less LP and move the price due to the unbalance created.

Impact

For this to have any impact the amount deposit must be significant in relation to the current pool balances. The loss and potential profit for the curator is dependent on how large the resulting unbalance is.

Code Snippet

https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/zaps/CurveLiquidityZap.sol#L127

Tool Used

Manual Review

Recommendation

One solution is to query the pool and make sure the depositing ratio is not more than x % from the current pool ratio.

Discussion

petarcalic99

Thank you for the report, its a valid finding however we thought about it already and there is no elegant solution. If the curator is allowed to lose 5% on the addLiquidity, he can do it in a loop multiple times. So one of the solutions we prepared is to add a small delay on the curve pool actions. However it will constrain the curator and we don't have the infra to monitor those at the moment so we will start without it and potentially add it later on. We also plan to add a whitelisting of the price range that he can enter a pool.

Issue L-9: `MetaVaultWrapper.decreaseRedeemRequest` treats wrapper shares as `1:1` to `AmphorVault` shares, breaking accounting and locking user funds

Source: https://github.com/sherlock-audit/2025-09-spectra-metavaults/issues/17

Summary

When `MetaVaultWrapper.decreaseRedeemRequest` is called, the `shares` argument specifies wrapper shares. However, when the request is forwarded to `AmphorVault`, the same `shares` argument is used (failing to use `_previewUnwrap` to calculate correct `AmphorVault` shares):

```
// Update pending balance
A.epochs[eid].redeemRequestBalance[msg.sender] = pending - shares;

// Decrease request on infra vault
IAmphor(W.infraVault).decreaseRedeemRequest(shares);
```

Root Cause

Using wrapper `shares` in `AmphorVault` without unwrapping them: https://github.com/sherlock-audit/2025-09-spectra-metavaults/blob/7c6eb4ff4b52d183fd6efee9d813c834dffa0cd5/metavaults-V1/src/MetaVaultWrapper.sol#L557

Internal Pre-conditions

Virtual supply of AmphorVault is different from virtual supply of the wrapper.

External Pre-conditions

None.

Attack Path

Happens by itself every time `decreaseRedeemRequest` is called with internal-preconditions.

Example 1. If AmphorVault virtual supply is larger than wrapper virtual supply:

- `totalVirtualInfraVaultSupply` = 100
- `totalVirtualSupply` = 40
- User requests to redeem 20 wrapper shares (this creates amphor vault redeem request of 50 amphor shares)

- User decreases redeem request by 10 wrapper shares: this gives 10 wrapper shares back to user, and decreases amphors redeem request by 10 amphor shares
- Amphor vault is settled
- wrapper redeems 40 amphor shares
- when user redeems, he is given assets worth `_previewUnwrap(10) = 25` amphor shares

A user is given correct amount of assets, but the vault now has less amphor shares than it should, breaking further accounting. For example, next users will be unable to redeem, because the wrapper won't have enough amphor shares to redeem.

Example 2. If AmphorVault virtual supply is less than wrapper virtual supply:

- `totalVirtualInfraVaultSupply` = 40
- `totalVirtualSupply` = 100
- User requests to redeem 50 wrapper shares (this creates amphor vault redeem request of 20 amphor shares)
- User decreases redeem request by 20 wrapper shares: this gives 20 wrapper shares back to user, and decreases amphors redeem request by 20 amphor shares
- Amphor vault is settled
- wrapper redeems 0 amphor shares
- when user redeems, he should be given assets worth `_previewUnwrap(30) = 12` amphor shares
- the transaction either reverts (since wrapper didn't redeem anything but still owes to user), or it succeeds, but reduces wrapper's asset token balance so that there are not enough funds for the later users to redeem.

A user redeem request either reverts (if amphor didn't have enough funds) or it works, but later users will not be able to redeem.

Either way, accounting is broken (wrapper will have non-matching amount of amphor shares)

Impact

- user `decreaseRedeemRequest` can revert in some situations
- internal accounting is broken
- funds of some users will be locked in the contract due to mis-accounting.
- wrapper has incorrect amount of amphor vault shares

Note: Since in current implementation `totalVirtualInfraVaultSupply = totalVirtualSupply` always, the issue described can not happen, but if/when this ratio is anything other than 1, the issue would become high.

Mitigation

`_previewUnwrap` shares before using with the amphor vault functions. Note: make sure unwrap is called on total remaining shares rather than delta shares, since `unwrap(share2 - share1) != unwrap(share2) - unwrap(share1)` due to rounding.

Discussion

panprog

Fix Review: Unwrapping delta amount is incorrect due to rounding (as described in mitigation):

```
// Decrease request on infra vault
IAmphor(W.infraVault).decreaseRedeemRequest(_previewUnwrap(shares));
```

Example:

- position with 10 wrapper shares worth 15 amphor shares (conversion rate is 1.5)
- User requests to decreaseRedeem to 9 wrapper shares (delta = 1)
- unwrap(1) = 1, amphor position reduces by 1 to 14 amphor shares
- ..repeat the same 9 times
- now user position is closed, but amphor redeem request is only reduced by 10 amphor shares (instead of 15), so now user position is closed while wrapper still has 5 amphor shares redeem request.

The correct way is to unwrap remaining shares: `amphorSharesDelta = _previewUnwrap(pending) - _previewUnwrap(pending - shares);`

petarcalic99

Yes thanks for the catch, will implement this and push

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.