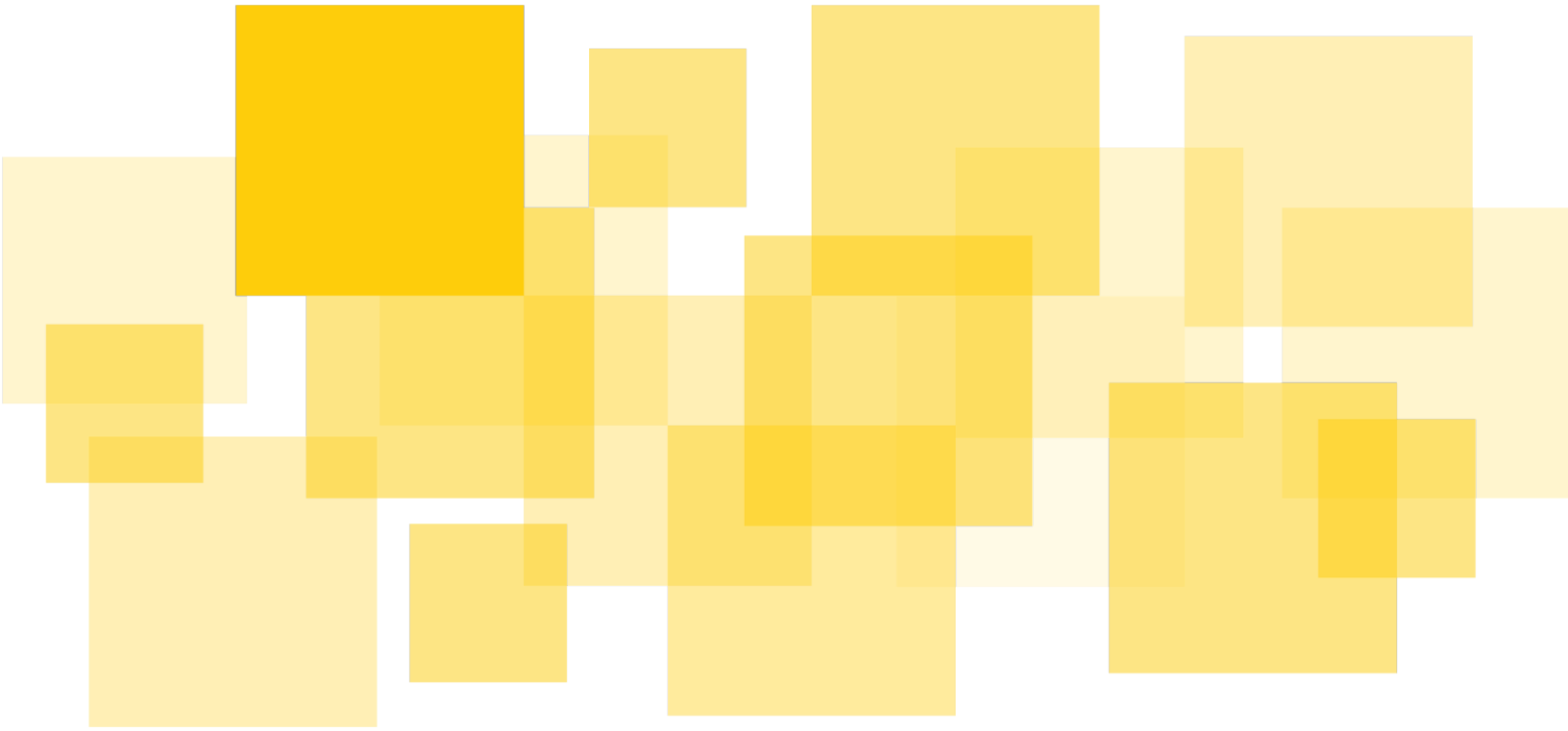
Security Audit Report

HydraDX Omnipool Withdrawal Fee

April 30, 2023



Prepared for HydraDX by Runtime Verification, Inc.

runtime verification

Contents

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of this contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk.

This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Summary

A bug was found recently that an attacker can get profits by manipulating prices of the tokens in the omnipool. The Hydrax team engaged Runtime verification in an audit related to the fixes of the bug.

Scope

Hydrax-node: pallets/omnipool/src

- base commit: f80300c1fed514761a6601f68a2f3afe6282d6f1
- frozen commit: cb53f0709aff736aea777d228fbc0cce99dc3dba

Hydrax-math: src/omnipool

- base commit: b459299bb4245514782d741faf028bb971a6f756
- frozen commit: 84632c3c7b71e16d53240e4bdc2adc693c941916

The review is limited in scope to consider only the code difference between two commits of the two repositories, though code of external libraries will be consulted whenever necessary. Non-chain and client-side portions of the codebase are not in the scope of this engagement.

Attack Explanation

The attack consists of 4 steps. We will use the `ETH-DAI` pair to illustrate the attack.

- **Step 1: the attacker swaps a large amount of ETH for DAI, and pushes the price of DAI high with respect to LRNA.**
- **Step 2: the attacker adds a large amount of DAI to the DAI pool when the DAI price is high.**
- **Step 3: the attacker swaps the DAI in step 1 back to ETH.**
  The swap has two sub-steps: the protocol first swaps DAI for LRNA in the DAI pool, and then it swaps LRNA for ETH in the ETH pool. The swapping from DAI to LRNA happens when the DAI pool has more liquidity than the DAI pool in the first step. As a result, the attacker can get more LNRA.
  For example, in a pool with 100DAI + 100LRNA, swapping 100DAI into the pool will get approximately 100 - 100*100/(100+100) = 50 LRNA out of the pool. However, in a pool with 10000DAI + 10000LRNA, swapping 100DAI into the the pool will result in approximately 10000 - 10000*10000/(10000+100) = 99 LRNA being swapped out of the pool. The excessive LRNA can be used to swap assets from other pools, in our example ETH.
- **Step 4: the attacker withdraws the liquidity from step 2.**

In order for this attack to be successful, the 4 steps should be done atomically. Otherwise, other protocol users can take advantage of this price manipulation.

Attack Mitigation

In order to mitigate the attack, the Hydrax team implemented 2 fixes:

- **Use price guardian when users add/remove liquidity.**
  When users add/remove liquidity to an asset pool, the protocol will check if the difference between the spot price and the external oracle price of the asset is within a predefined range around the oracle price. If the tolerance is exceeded, the add/remove liquidity functionality will be paused.
- **Charge withdrawal fee when users remove liquidity.**
  When users withdraw the liquidity from the asset pool, the protocol will charge withdrawal fee which is $max(min(\frac{|o-p|}{o}, 1), f_m)$ of the amount of the asset that will be withdrawn from the pool. o is the oracle price, p is the spot price and f_m is the minimal withdrawal fee. The larger the price difference is, the more withdrawal fee the protocol will charge.

The first fix tries to prevent step 2 and step 4 when price is being manipulated. The second fix reduces the profits from step 3 when the attacker removes the liquidity.

Protocol Implementation

Operation: Adding Liquidity- Single Asset Liquidity Provision

Implementation

Settings

Same as before

Preconditions

1. $amount \geq MinimumPoolLiquidity$
2. $amount = 0 \vee Balance_{asset}^{who} - \Delta Balance_{asset} \geq MultiCurrency::Accounts(who)(asset).frozen$
3. $asset \in Assets$
4. $StableCoinAssetId \in Assets$
5. $ADD\_LIQUIDITY \in tradable\ assets$
6. $\left|oracle_{asset\_id} - price_{asset\_id}\right| * 2 \leq (price_{asset\_id} + oracle_{asset\_id}) * maxDiffAllowed$

Post state

Same as before

Post condition

Same as before

Operation: Removing Liquidity

Implementation

Settings

$who = ensure\_signed(origin)$

$asset\_id = Positions(position\_id).asset\_id$

$Price_{position\_id} = Position(position\_id).price$

$Price_{asset\_id} = \frac{Value_{asset\_id}}{Balance_{asset\_id}}$ is the current price of the asset over the stable coin asset.

$\alpha = \frac{Price_{position\_id} - Price_{asset\_id}}{Price_{position\_id} + Price_{asset\_id}}$, $\beta = 1 - \alpha$

$$\Delta b = [Price_{asset} < Price_{position}] * \lfloor \alpha * amount \rfloor$$

$$\Delta shares_{asset\_id} = amount - \Delta b$$

$x = \frac{|oracle_{asset\_id} - price_{asset\_id}|}{oracle_{asset\_id}}$, $f_w = max(min(x, 1), f_m)$, and f_m is the minimal withdrawal fee.

$$\Delta Balance_{asset\_id} = \lfloor \lfloor \frac{Balance_{asset\_id} * \Delta shares_{asset\_id}}{shares_{asset\_id}} \rfloor * (1 - f_w) \rfloor$$

$$\Delta Value_{asset\_id} = \lfloor \lfloor \frac{\Delta Balance_{asset\_id} * Value_{asset\_id}}{Balance_{asset\_id}} \rfloor * (1 - f_w) \rfloor$$

$$\Delta imbalance = [\Delta Balance_{asset\_id} \neq 0 \wedge Imbalance_{LRNA}.value \neq 0 \wedge Balance_{LRNA} \neq 0] * Increase(\lfloor \lfloor \frac{(Value_{asset\_id} - \Delta Value_{asset\_id}) * Imbalance_{LRNA}.value}{Balance_{asset\_id} - \Delta Balance_{asset\_id}} \rfloor * \frac{\Delta Balance_{asset\_id}}{Balance_{LRNA}} \rfloor)$$

$$\Delta Balance^{who}_{LRNA} = [Price_{asset\_id} > Price_{position\_id}] * \lfloor Price_{asset\_id} * (\lfloor \beta * \Delta Balance_{asset\_id} \rfloor - \Delta Balance_{asset\_id}) \rfloor$$

Preconditions:

1. $NFT_{omnipool}(position\_id) = who$
2. $position\_id \in Positions$
3. $Position(position\_id).shares \geq amount$
4. $StableCoinAssetId \in Assets$
5. $asset\_id \in Assets$
6. $REMOVE\_LIQUIDITY \in tradable\ asset$
7. $Balance_{asset\_id} - \Delta Balance_{asset\_id} \geq MultiCurrency::Accounts(Omnipool)(asset).frozen$
8. $Balance_{LRNA} - \Delta Value_{asset\_id} \geq MultiCurrency::Accounts(Omnipool)(LRNA).frozen$
9. $|oracle_{asset\_id} - price_{asset\_id}| * 2 \leq (price_{asset\_id} + oracle_{asset\_id}) * maxDiffAllowed$

Post state

Same as before

Post condition

Same as before

Findings

1. Prices used in the `ensure_price` function are inconsistent with design intention.

[Severity: Informative | Difficulty: Easy | Category: Security]

Issue: The `ensure_price` function is to check if the difference between the spot price and the oracle price is within a certain range. However, depending on which price is higher, checking $\left|oracle_{asset\_id} - price_{asset\_id}\right| \leq price_{asset\_id} * maxDiffAllowed$ and $\left|oracle_{asset\_id} - price_{asset\_id}\right| \leq oracle_{asset\_id} * maxDiffAllowed$ may yield different results.

Recommendation: use $min(price_{asset\_id}, oracle_{asset\_id})$ or $average(price_{asset\_id}, oracle_{asset\_id})$ in the right hand side of the inequality.

Status: fixed in PR#567