Exploiting aToken liquidity addition in stableswap - post mortem

Updated July 7, 2025 • 7 min read

Jakub Panik

Summary

On June 18th 2025 at 2:48 PM CEST, the Intergalactic team received a report via the Hydration Immunefi bug bounty programme, disclosing a critical vulnerability in the transfer function of aTokens, which could have potentially enabled holders of aTokens (i.e. tokens deposited on the Hydration Money Market) to mint more stablepool shares than they would have been entitled to.

Within 2 hours after the submission, the Intergalactic team managed to evaluate the report, having acknowledged the severity of the vulnerability, the Technical committee performed liquidity addition pause for GDOT pool, which was the only affected pool in question. Securing the funds at risk.

Immediately after this, an emergency runtime upgrade was proposed on Hydration OpenGov. Due to the sensitive nature of the issue, the upgrade was proposed in "stealth", i.e. the code changes were not published on GitHub until its enactment at the same time as the liquidity add pause was proposed. (you can see them now here). Approximately 7 hours after the report, the fixed Hydration runtime was applied on the mainnet, and the transaction pause was removed by the Technical committee.

Given the severity of this vulnerability, the IGL team has prepared a list of measures which will help improve the security posture of the development process, with the goal of minimizing the probability that similar vulnerabilities get introduced in the future.

According to our estimates, the impact of the vulnerability - which remained unexploited - could have led to up to a $22M loss for the Protocol and LPs in Stablepools with aTokens. In accordance with the rules of the Hydration Immunefi bug bounty programme, this is a Critical issue and the whitehat is entitled to the maximum payout of $500,000 in HDX.

To avoid sudden price pressure on HDX, it was negotiated with the whitehat, that the payout is done in 2 parts: $250,000 in HDX vested for 20 months, and $250,000 in a stablecoins. This is still subject to ratification by Hydration OpenGov.

The vulnerability

Hydration uses EVM version of AAVE money market integrated into its Substrate chain allowing users to lend and borrow tokens across both ERC-20 and Substrate-native assets. AAVE introduces the concep aTokens, yield-bearing assets which are rebasing (The token amount in wallets change over time). This

Contents

behavior led to errors in calculation in some scenarios on Hydration, such as when converting between native tokens and aTokens, or when transferring aTokens from one account to another. As a result, for some cases the Runtime was unable to transfer the full aToken balance of a user, leaving behind small residual amounts ("dust").

Since ERC20 has no concept of existential deposit (ED), this dust is never automatically cleaned up. Over time, this could lead to increased gas usage, performance degradation, and confusing balance displays.

To address this, following issue was opened:

https://github.com/galacticcouncil/hydration-node/issues/1092

Initially, the plan was to fix the issue within the Hydration route executor implementation, but later the decision was made to solve it in the `Currencies` pallet's transfer function instead, providing a generic solution for all aToken transfers.

The implemented solution was the following: when transferring aTokens, if the remaining balance after the transfer is less than the ED, the Runtime would perform an `AAVE withdraw all` on behalf of the recipient, ensuring that all aTokens are withdrawn, sent to the recipient, and no dust remains on the origin account. To calculate whether the remaining balance was below ED and trigger a transfer including `AAVE withdraw all`, the following line was introduced to the Runtime (among others):

```
...
let diff = atoken_balance.saturating_sub(amount);
...
```

Source code: hydration-node/runtime/hydradx/src/evm/aave_trade_executor.rs at c16eeb7c

The problem is that this line uses **saturating math**, meaning even if the user's balance is lower than the transfer amount, the subtraction does not panic or fail, it just returns zero. As a result, the system would execute an `AAVE withdraw all` on behalf of the recipient **even when the sender had insufficient balance**, leading to a successful call regardless.

On its own, this might not have posed a critical risk, however, since this change was made at the **generic transfer level**, it caused unintended **side effects** across all parts of the system that rely on Hydration's transfer logic, resulting in a scope creep.

The most critical affected component was the `Stableswap::add_liquidity_shares` extrinsic, which mints liquidity shares for the user in exchange for a user-provided asset. At the end of this extrinsic, two key actions take place: minting the user-specified amount of shares and transferring the corresponding amount of the asset.

In a potential exploit scenario, a user could call the extrinsic with an aToken amount greater than their actual balance. Because the transfer logic no longer failed when the user had insufficient funds, the extrinsic executed successfully: it minted the user-specified number of shares and attempted to transfer the asset. Since we had a missing balance check in aToken transfer logic, the transfer did not fail, resulting in an invalid share minting that was not backed by real assets.

Mitigation

To mitigate the vulnerability described above, we disabled the fix for the aToken rounding issue. We will be further working on proper fix of the rounding issue that will be audited.

Follow-up Measures

1. Avoid Saturating Math by Default

- Saturating arithmetic (e.g. `saturating_sub`) should **not** be used by default, as it can silently hide critical errors like underflows.
- In this incident, the use of `saturating_sub` allowed a transfer to proceed even when the sender had insufficient balance, leading to unintended consequences.

Actions: Default to `checked_*` math operations, use saturating math **only when explicitly required** and accompanied by comments explaining the rationale.

We will be implementing a code watcher that will automatically check for the usage of this math in the proposed PRs.

2. Improve Property-Based and Edge Case Testing

- The modified transfer logic lacked rigorous property tests covering edge conditions.
- Original unit tests were not properly adapted to cover changes introduced by the new transfer behavior.

Actions: We will be Implementing property-based tests and ensure that our tests explicitly differentiate between saturating and checked math behaviour.

3. Strengthen Integration Tests

- Current integration tests are too far from real-world mainnet scenarios.

Actions: Improve test environments to more closely reflect mainnet dynamics, and create testing guidelines that enforce this.

4. Stricter Asset Balance Checks

- Internal logic trusts other pallets (e.g. EVM subsystem, aTokens) too much, without validating balances.

Actions: Be less trusting of external pallet behaviour. Introduce explicit sanity checks and asserts for balances and asset movements in high-impact logic.

5. Circuit Breakers

- Implementing a token minting limit per asset to enable automatic timed asset lockdown functionality in case of critical scenario.

Actions: Add safeguards that limit minting and burning of excessive amount of tokens per time period. Limit thresholds should be **calibrated based normal chain actions and on bug the bounty**, making it always more plausible for a hacker to report the issue than to exploit it.

6. Better Pull Request Classification

- This issue stemmed from a seemingly small change with wide-ranging consequences.

Action: Label PRs based on their potential scope and subsystem integrations (e.g. AAVE, EVM, Stableswap). Assign security audit or extra review automatically based on label and **require impact analysis and mitigation plan for PRs affecting shared or critical components.**

7. Extra Caution Around EVM Subsystem

- EVM-related integrations and contracts carry higher risks.

Actions: Introduce stricter review and testing requirements for any code interacting with the EVM, including multi-system transfer logic.

8. Fuzz Testing with Mainnet-Like Setup

- Fuzzing was missing or insufficient for the affected feature.

Actions: Integrate fuzzing as part of the dev and testnet lifecycle of every new feature, ensure fuzzing covers realistic state and liquidity levels, Introduce invariants into the code that validate crucial business logic and allow the fuzzer to trigger a crash.

9. More Frequent and Continuous Audits

- The fix introducing the issue was not audited as one of the few components in Hydration even though it touched critical transfer function.

Actions: Double down on auditing and continuous audits of features

Moving forward

While the response time from receiving the bug report notification to protecting the funds was under 2 hours, whole Intergalactic team is committed to improving the internal standards and is committed to not let this happen in the future. As such, we have implemented or started implementing all of the points above.

Nevertheless this shows how important the Immunefi bug bounty programme is as the final line of defence. We are grateful for the work of all of the whitehats looking at the code and reporting issues as this, together with good security practices and continuous audits, helps keep the Hydration chain secure.

More from this blog

Hydration oracle manipulation - post mortem

On the 3rd of July 2024 we have received a report in the Immunefi platform from top hacker on the platform labeled as “Critical” called “Risk free oracle manipulation”. and it looked as following: Hydra is an interesting substrate based protocol tha...

Mar 7, 2025 13 min read

Snek Stall Post Mortem

How Basilisk stopped at block #2 145 599 and how we'll fix it

Nov 9, 2022 5 min read

Hyperspace

3 posts published