Hydration node

Security Review

Cantina Managed review by:

0xLeastwood, Lead Security Researcher
Zigtur, Security Researcher

April 7, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Hydration is a next-gen DeFi protocol which is designed to bring an ocean of liquidity to Polkadot.

From Nov 28th to Dec 1st the Cantina team conducted a review of hydration-node on commit hash 9588c486. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	4	2	2
Gas Optimizations	0	0	0
Informational	3	3	0
Total	**8**	**6**	**2**

3 Findings

3.1 Medium Risk

3.1.1 Liquidation is not executable when debt asset is collateral asset

Severity: Medium Risk

Context: lib.rs#L233-L240, lib.rs#L457-L467

Description: In AAVE V3, a borrower can provide multiple assets as collateral and borrow these same assets. For example, a user may deposit 20 USDC and other assets to borrow 100 USDC. Then, a liquidator should be able to liquidate the position when passing the same asset as debt and collateral.

However, the collateral asset and the debt asset are expected to be different in the `liquidate` function. It executes a swap which does not support having the same input and output assets.

Such liquidation attempt will fail, the `T::Router::sell` call will return an `Error::<T>::NotAllowed` triggered from `RouteExecutor::do_sell`.

```
fn do_sell(
    origin: T::RuntimeOrigin,
    asset_in: T::AssetId,
    asset_out: T::AssetId,
    amount_in: T::Balance,
    min_amount_out: T::Balance,
    route: Vec<Trade<T::AssetId>>,
) -> Result<(), DispatchError> {
    let who = ensure_signed(origin.clone())?;

    ensure!(asset_in != asset_out, Error::<T>::NotAllowed); // @POC: Debt can't be Collateral.
```

Recommendation: The swap logic should not be executed when the debt asset is also the collateral asset.

Hydration: Fixed in commit 4d443035.

Cantina Managed: Fixed. Recommendation has been applied. The swap is executed if the debt asset and the collateral asset are different.

3.2 Low Risk

3.2.1 Incorrect slippage protection allows an external party to make profits on liquidations

Severity: Low Risk

Context: lib.rs#L233-L240

Description: During a liquidation, the collateral earned is swapped to the debt asset. After this swap, a check is executed to ensure that the debt asset earned is positive meaning the liquidation is profitable.

However, the swap (i.e. the logic in `T::Router::sell`) is not protected against slippage. The `min_amount_out` parameter allowing slippage protection is hardcoded to `1`, deactivating the slippage protection. This can be leveraged by an attacker to reduce the profit made by the liquidation and extract value from it.

The attacker would use a sandwich attack pattern:

1. Minimise liquidation profit by manipulating the balance of pool reserves.
2. Protocol executes the liquidation, making non-zero but negligible profit.
3. Swap assets back through the pool to recover the initial price, stealing part of the profit that the liquidation should have made.

```
pub fn liquidate(
    origin: OriginFor<T>,
    collateral_asset: AssetId,
    debt_asset: AssetId,
    user: EvmAddress,
    debt_to_cover: Balance,
    route: Vec<Trade<AssetId>>,
) -> DispatchResult {
    // ...

    // swap collateral asset for borrow asset
    let collateral_earned = <T as Config>::Currency::balance(collateral_asset, &pallet_acc)
        .checked_sub(collateral_asset_initial_balance)
        .ok_or(ArithmeticError::Overflow)?;
    T::Router::sell(
        RawOrigin::Signed(pallet_acc.clone()).into(),
        collateral_asset,
        debt_asset,
        collateral_earned,
        1,                  // @POC: hardcoded `min_amount_out` makes slippage protection inefficient
        route.clone(),

    // ...
}
```

Note: this issue is unlikely to occur because the attacker will make more profit by directly liquidating the position.

Recommendation: The `T::Router::sell` function should be called with a correct `min_amount_out` value to ensure that the slippage mechanism protects the profit.

Moreover, an option to receive the debt asset and avoid swapping to the collateral asset could be implemented to ensure that a liquidation is still possible when an attacker manipulates the pool's price.

Hydration: Acknowledged.

Cantina Managed: Acknowledged. A party would make more profit from liquidating directly through the EVM than from sandwiching a liquidation, making this issue unlikely to occur.

3.2.2 Hardcoded gas limit may break liquidations

Severity: Low Risk

Context: lib.rs#L221, assets.rs#L1589

Description: The `liquidate` extrinsic creates an EVM call to the money market contract to liquidate a position. This EVM call is built with a `1_000_000` gas limit.

```
pub const LiquidationGasLimit: u64 = 1_000_000;
```

This hardcoded 1 million gas limit lead liquidations that require more gas to revert due to an out-of-gas error. A user can increase the gas required to liquidate their position by supplying multiple collaterals and borrowing multiple debt assets. This will make account health calculations more gas consuming which may break the 1 million gas limit.

Note: This issue requires the money market contract to support a high number of assets for the user to be able to exploit it. Upon investigating a number of AAVE liquidations on mainnet, the gas cost for individual oracle price queries are in the range of 25-60k. On the lower end, this would require a total of 40 collateral/debt assets but realistically it would be considerably less than that as there is a base cost for other components of the liquidation and the median gas cost for oracle price queries is realistically higher.

Recommendation: The gas limit set for the liquidation EVM call could be passed as input to the `liquidate` extrinsic. This will allow supporting new assets in the money market contract while still being able to liquidate through this extrinsic. Note that this input value should still be capped to a reasonable value. For example, a 10 million gas cap could be checked.

Hydration: Fixed in commit 4d443035.

Cantina Managed: Fixed. The liquidation gas limit has been set to 4 million gas which is a reasonable gas limit value considering the number of supported assets.

3.2.3 Liquidation can be avoided by manipulating the swap pool

Severity: Low Risk

Context: lib.rs#L248-L251

Description: The current `liquidate` extrinsic makes a strong assumption on the asset prices. The liquidation price and the pool price must be close one to the other.

A liquidation through this extrinsic needs to be profitable to ensure that the minted debt assets are burnt after the liquidation and the swap. If it is not profitable, the whole transaction is reverted and the position is not liquidated.

However, the liquidation and swap operations use different price sources. Even worse, a user is able to manipulate one of this price source (i.e. the swap pool) especially in case of low liquidity in this pool. The discrepancies in prices may lead the liquidation to not being executable as not being profitable.

Recommendation: By design, the pool price and the oracle price can be different.

Hydration: Acknowledged, this is to be expected. As protocol worst case scenario for on chain liquidation is that whole profit is spent on slippage and swap fees - which is still great to our LP's and stakers, so if you manipulate price just so the protocol doesn't liquidate you essentially earn protocol 2x of that (assuming you will move the price back after liquidation).

Cantina Managed: Acknowledged. This issue is unlikely to happen as liquidations are still executable through the EVM.

3.2.4 Multi-hop swaps are not supported

Severity: Low Risk

Context: lib.rs#L185-L194

Description: On liquidation, the `liquidate()` function swaps all collateral earned into the target `debt_asset` through the AMM. The route is strictly defined as follows:

```
// `route` needs to be provided in order to calculate correct TX fee.
// Ensure that the provided route matches the route we get from the router.
ensure!(
    route
        == T::Router::get_route(AssetPair {
            asset_in: collateral_asset,
            asset_out: debt_asset,
        }),
    Error::<T>::InvalidRoute
);
```

Effectively there is only a single permissible path, i.e. `collateral_asset -> debt_asset`. However, it is likely that other swap paths provide a better price for swap execution. Introducing intermediary hops allows the liquidator to take advantage of pools with deeper liquidity. For example, the swap path `collateral_asset -> intermediary_asset -> debt_asset` interacts with two pools, namely; `(collateral_asset, intermediary_asset)` and `(intermediary_asset, debt_asset)`..

Recommendation: Consider only checking that the entry and exit assets are `collateral_asset` and `debt_asset` respectively.

Hydration: Fixed in commit 4d443035.

Cantina Managed: Fixed. Recommendation has been applied by removing the route check. The entry and exit assets are already checked during the execution of `T::Router::sell` thus they do not require to be double checked in `liquidate`.

3.3 Informational

3.3.1 Overflow error is used instead of underflow error

Severity: Informational

Context: lib.rs#L230-L232, lib.rs#L244-L246

Description: The `debt_asset_earned` and `collateral_earned` are calculated from the final balances subtracted by the initial balances. However, when the subtraction fails due to underflow, the `liquidate` function will return an `ArithmeticError::Overflow` error.

Recommendation: Consider replacing the `ArithmeticError::Overflow` error with the `ArithmeticError::Underflow`.

Hydration: Fixed in commit 4d443035.

Cantina Managed: Fixed. The collateral calculation now uses the `ArithmeticError::Underflow` error and the debt calculation uses the `NotProfitable` error.

3.3.2 Route variable is cloned to call `Router::sell`

Severity: Informational

Context: lib.rs#L233-L239

Description: During the call to `T::Router::sell`, the `route` variable is cloned and this clone is passed to the function. However, the `route` variable is not used in `liquidate` after this call. As the variable is not used anymore, the clone is not needed and the variable ownership can be transferred. `T::Router::sell` can be called with `route` instead of `route.clone()` to avoid unnecessary computations.

Recommendation: Remove the `.clone()` code to avoid unnecessary computations.

```
T::Router::sell(
    RawOrigin::Signed(pallet_acc.clone()).into(),
    collateral_asset,
    debt_asset,
    collateral_earned,
    1,
    route,
```

Hydration: Fixed in commit 4d443035.

Cantina Managed: Fixed. The recommendation has been applied.

3.3.3 Money market contract address is hardcoded to testnet address

Severity: Informational

Context: assets.rs#L1587-L1588

Description: The liquidation pallet uses the `MoneyMarketContract` variable. This is initialized to the `0xf550bcd9b766843d72fc4c809a839633fd09b643` address, which corresponds to the testnet deployment. The mainnet deployment for this contract is at address `0x1b02E051683b5cfaC5929C25E84adb26ECf87B38`.

Recommendation: The configuration should select the correct money market contract address depending on if the configuration is mainnet or devnet.

Hydration: Fixed in commit d56c7b2e.

Cantina Managed: Fixed. The contract address is not hardcoded anymore. It is now a storage variable that can be modified through an extrinsic. This new extrinsic ensures only the `root` origin can call it.