Volt System Oracle Security Review

AlwaysBeGrowing

Reviewer
bookland.eth | russell

August 2, 2022

1 Executive Summary

I spent a total of 20 hours reviewing the changes split across four days. I have reviewed Volt protocol in the past so I already had a good understanding of the system.

In this review, I did an in depth review of VoltSystemOracle.sol VoltSystemOracle.t.sol. I also did an in depth review on the deployment scripts and integration tests. The review was over PR 82. Specifically, commit 73a7ead.

I found one issue with the implementation that was fully fixed. There are a few limitations to be aware of all outlined in VIP2.md.

PR	Commit
PR 82	73a7ead

Summary

Timeline	July 11, 2022 - July 14, 2022
Methods	Manual Review
Documentation	Good
Testing Coverage	High

Total Issues

Critical Risk	0
High Risk	1
Medium Risk	0

Contents

2 Always Be Growing

ABG is a talented group of engineers focused on growing the web3 ecosystem. Learn more at https://abg.garden

3 Introduction

Volt is an inflation resistant stablecoin. In this PR they are swapping from using a CPI oracle to using a Volt System Oracle. The Volt System Oracle is configured with a starting price, a start period of when interest should start accruing, and the monthly price change denominated in basis points. Once the Volt System Oracle has been deployed these values can not be changed. For any changes of these values to happen a new oracle would need deployed and Volt protocol would need configured to use the new oracle.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a one person team. Any modifications to the code will require a new security review.

4 Findings

4.1 High Risk

4.1.1 Role escalation possible from guardian to govenor

Severity: High

Context: `OptimisticTimelock.sol#L30-L32`

Description: The optimistic timelocks have the governor role. A user with the guardian role can call the `becomeAdmin()` method on the timelock to become the admin, they would then be able execute transactions as if they had the governor role.

```
    function becomeAdmin() public onlyGuardianOrGovernor {
        this.grantRole(TIMELOCK_ADMIN_ROLE, msg.sender);
    }
```

Recommendation: Deploy new timelocks without the ability for guardians to grant themselves the `TIMELOCK_ADMIN_ROLE`

Volt: Fixed by deploying new timelocks on mainnet and arbitrum.

ABG: Resolved.

5 Methodology

I spent around half of my time reviewing `VoltSystemOracle.sol` & `VoltSystemOracle.t.sol` & the second half reviewing deployment and governance logic. Below is a non-exhaustive list of what was tested and explored.

1. High level protocol overview and line by line code review. I looked through each of the files updated in this PR to get a full understanding of the changes and to decide where I would spend the majority of my time.
2. Manual verification of the `getCurrentOraclePrice()` and `compoundInterest()` functions. I wanted to ensure that these functions were fully and correctly tested.

   Some specific things I looked into:

   - Are the `_calculateDelta` and `_calculateLinearInterpolation` helpers in `VoltSystemOracle.t.sol` correct?
   - Does `getCurrentOraclePrice()` work correctly within a specific period?
   - Does `getCurrentOraclePrice()` work correctly across periods?
   - Any situations where `getCurrentOraclePrice()` could revert?
   - Does the value returned by `getCurrentOraclePrice()` work correctly if there are multiple time periods between each `compoundInterest()` call?
   - How long into the future can this oracle be depended on?
3. I explored the impact of what would happen if `compoundInterest()` did not get called for multiple days. This situation had already been called out and explored by the core team and they will be using a keeper to prevent long periods of time without a `compoundInterest()` call. As outlined by the team, calling this method in a timely fashion is important to prevent a possible price arbitrage between before the `compoundInterest()` call and after the call. Even in the case of a black swan event where `compoundInterest()` does go multiple days without being called, the arbitrage

opportunity is minimal due to the mint/redeem limits & fees.

I ran some rough numbers on what the impact would be if there was an issue with the keeper and `compoundInterest()` is not called for 24h. Exploiting this on Arbitrum would not make sense unless there was a much longer period because there is a 5 basis fee on both mint and redeem that would be larger than any potential profit. For mainnet USDC PSM it there's currently a 0 basis fee on redeem and will soon be a 0 basis fee on mint, making this theoretically exploitable. There is a rate limit on how much volt can be minted/redeemed that significantly reduces the possible profit from this.

In the current configuration, the max amount that can be minted at once is 10m, and if flash loaning 10m (dydx w/ 0% fee) to atomically arb this at 20 monthly basis points and 1 day of delay to `compoundInterest()` then possible arb profit would be ~$600.

Might be worth considering to keep a very small fee on mint or redeem if ever increasing the mint cap in the future

4. Examination of all privileged roles in the system. I looked through each of the roles and addresses to ensure that the addresses being used in the code matched the addresses that were on mainnet and arbitrum. This included checking the timelock roles, core roles, and the passthrough oracle owner. All roles match what was expected. Additionally, there is comprehensive role based fork testing ensuring that the roles match what is expected and that no unexpected addresses have been granted a role.
5. Walk through of deployment logic. I examined the current governance deployment framework and brainstormed any ways where it could go wrong. The main areas of concern would be a misconfiguration of the new system, or a difference in price between the new price oracle and the old price oracle causing a possible arbitrage opportunity. Both of these risks have been mitigated through comprehensive integration testing & temporarly pausing minting until the pricing oracle has been updated.

6 Test Quality

The Volt team has done a high level of diligence in ensuring that the code is fully tested and reliable. They used a combination of unit test, fuzz tests, and integration tests.

The new volt system oracle has been comprehensively tested through fuzz and unit tests that reimplement the LERP algorithm. These ensure that the oracle correctly follows the LERP algorithm at any point in time and that compounding works as expected. Additionally, there are detailed mainnet fuzz tests that simulate the execution of the proposal and ensure the `mintAmountOut()` and the `redeemAmountOut()` functions will return the correct amounts after the upgrade.

There is also role based testing that makes sure all roles are set up as expected, and a simulation framework that gives a high level of assurance that the governance proposal executes correctly.

7 Recommendations

- Magic strings (addresses) are used a few places in the code. I recommend naming all addresses that are being used in the code. This would help reviewers and devs to understand the significance of each address. This would also help prevent against regression issues in the future where an address is changed only in one place but not in others.
- I recommend exploring simplifying and clarifying the deployment logic and integration tests. While the current logic appears to be sound and well tested there seems to be some complexity which increases the risk of an error in future deployments. This would also simplify deploying to additional networks in the future.
- Consider having a very small bips fee for minting/redeeming volt to negate any potential profit from flash mint/redeems. This is to remove/reduce any possible profit from #3 outlined above.