yAudit USDM Delayed Oracle Review

Review Resources:

None beyond the code repositories.

Auditors:

- adriro
- Jackson

Table of Contents

Review Summary

USDM

USDM is a daily rebasing yield-bearing stablecoin created by the Mountain Protocol. yAudit was asked to audit a DelayedERC4626Oracle, which is intended to delay the price propagation of an underlying ERC4626 vault. This was done due to the fact that ERC4626 vaults are susceptible to donation attacks and the oracle was thought of as a potential mitigation to this attack.

The DelayedERC4626Oracle was reviewed over 1 day. The code review was performed by 2 auditors between June 14, 2024 and June 15, 2024. The repository was not under active development during the review, and the review was limited to the latest commit at the start of the review. This was commit 9f593a9d729fbbcc5be9e360102c264add26c41b of the Steakhouse Financial repo.

Scope

The scope of the review consisted of the DelayedERC4626Oracle contract.

After the findings were presented to the Mountain Protocol team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Mountain Protocol and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Low	None of the functions are access controlled. Due the the sensitive nature of the contract the authors should consider access control.
Mathematics	Good	There is very little mathematics in the contract.
Complexity	Good	There is little complexity in the implementation. However there are security nuances due to the sensitive nature of the contract.
Libraries	Good	The contract makes use of the OpenZeppelin ERC4626 interface.
Decentralization	Good	The update function is fully accessible to the public.
Code stability	Good	No changes were made the code while the audit was performed.
Documentation	Low	No documentation was provided for the oracle.
Monitoring	Low	None of the functions emit events.
Testing and verification	Good	There were both unit and integration tests.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - Delayed prices can be manipulated

Technical Details

The purpose of the `DelayedERC4626Oracle` is to render single-block donation attacks infeasible. However, it is still possible to manipulate the delayed price. Consider an attack where the attacker manipulates the price of the underlying vault, calls `update()` to set the delayed price, then puts the price of the underlying vault back. The attacker then waits until the appropriate delay has occurred and uses the manipulated price in their attack.

Impact

Medium. The purpose of the delayed oracle is to decrease the risk of such an attack. However, since there is no way to override a manipulated price once it has been set, an attacker could still use the manipulated delayed price to perform an attack.

Recommendation

Make the `update()` function protected such that only trusted addresses can call it, or implement a protected emergency function to override a manipulated price.

Developer Response

Our understanding is that the only way to manipulate the wUSDM conversion price is to increase the amount of USDM in the contract. This can be done by compromising the rebase capacity of USDM or “gifting” USDM to wUSDM. In both cases, the price should reflect this change. What this contract prevents is that the attacker can’t use the new price during the same block. Therefore he can’t be sure to be the one profiting from the attack. This makes the attack less likely.

It is important to ensure that anyone can update the price so there is no trust assumption on some people operating this contract.

Low Findings

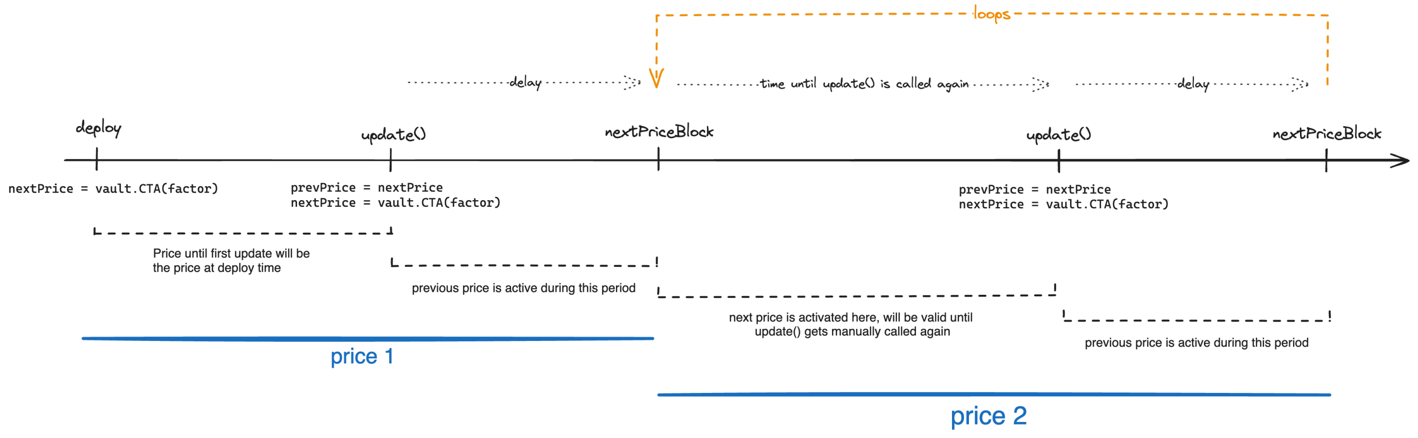
1. Low - Lack of staleness check

The Oracle relies on manual price updates, which could potentially lead to the reporting of stale prices if the updates are not performed regularly or in a timely manner.

Technical details

The DelayedERC4626Oracle.sol contract operates by caching the previous price on each price update. The cached price is active until the configured delay is hit, at which point the most recently fetched price becomes the active one.

This update process is manual, the Oracle relies on explicit calls to the `update()` function to refresh the price. Once the most recently fetched price becomes active, it will remain valid indefinitely until the `update()` function is called again. Even after the `update()` function is called, the price will continue to be valid until the next delay period has elapsed.


loops
delay
time until update() is called again
delay
deploy
update()
nextPriceBlock
update()
nextPriceBlock
nextPrice = vault.CTA(factor)
prevPrice = nextPrice
nextPrice = vault.CTA(factor)
prevPrice = nextPrice
nextPrice = vault.CTA(factor)
Price until first update will be the price at deploy time
previous price is active during this period
next price is activated here, will be valid until update() gets manually called again
previous price is active during this period
price 1
price 2

Here represented by the duration between `nextPriceBlock` and `update()`, the validity period for the cached price can grow indefinitely long, as there is no guarantee about when the next call to `update()` will happen.

Impact

Low. The impact would depend on the context where the Oracle is used. For example, if used in a lending market, it could lead to invalid borrows or liquidations. However, since the Oracle is expected to be used for wUSDM, it would likely require a long time (under normal circumstances) for the staleness to be meaningful, as USDM is a stablecoin with a slow rebase multiplier that is triggered at a daily rate.

Recommendation

One possibility would be to implement an internal check when the price is fetched. This would conflict with the ERC4626 standard, as `convertToAssets()` is not allowed to revert. Additionally, this approach still requires a design discussion about the behavior of `update()` when the current price is stale, specifically whether it should update the price immediately or still delay the update.

Another alternative could be to store the timestamp for the prices and report the time of the active one, letting users of the Oracle handle the situation.

Developer Response

This is acknowledged as a limitation but in the current state no other solution was found. This is why the `update()` function is permissionless allowing anyone to call it. Many parties have an incentive to keep the contract updated.

Baring TradFi interest rates at a level unheard of (>50%), a few days of delay in calling the update() function isn't a problem as most oracles used operate with a larger price error.

Informational Findings

1. Informational - Inflation attack at deployment time

The DelayedERC4626Oracle.sol contract is vulnerable to price manipulation during deployment, as the delay mechanism is not enforced in the constructor.

Technical details

When the DelayedERC4626Oracle.sol is constructed, its active price will be taken from the `nextPrice` variable that is initialized in the constructor.

```
    constructor(IERC4626 underlying_, uint256 delay_) {
        underlying = underlying_;
        delay = delay_;
        underlyingFactor = 10 ** underlying.decimals();
        nextPrice = underlying.convertToAssets(underlyingFactor);
    }
```

An attacker could inflate the value of wUSDM by front-running the deployment resulting in an immediate effective price, as the delay mechanism is only enforced during the call to `update()` and not when the contract is initialized.

Impact

Informational.

Recommendation

After deploying the Oracle, ensure it is in a safe state before enabling any market.

Developer Response

Acknowledged. The time to setup will be way higher than the delay.

2. Informational - Use time units to represent the delay

The implementation of DelayedERC4626Oracle.sol uses the block height to represent the delay when prices are updated.

Technical details

The `update()` function sets the delay as an offset to the current block number.

```
    function update() public {
        require(
            block.number >= nextPriceBlock,
            "Can only update after the delay is passed"
        );
        prevPrice = nextPrice;
        nextPrice = underlying.convertToAssets(underlyingFactor);
        nextPriceBlock = block.number + delay;
    }
```

Impact

Informational.

Recommendation

Consider using time units to represent the price update delay, using a metric that is independent of the block production rate. This would also help to standardize deployments across different chains.

Developer Response

Fixed in https://github.com/Steakhouse-Financial/delayed-oracle/pull/1

Final Remarks

Since USDM has already been deployed and is a rebasing token, it is difficult for Mountain Protocol to fix the donation attack problem with an upgrade of the contracts. They therefore elected to implement this delayed price oracle instead. Otherwise, it would have been suggested that Mountain Protocol fixed the underlying cause of the donation attack in USDM directly.

A few other observations about the oracle were made, and their severity is determined by the selection of the delay value during deployment as well as the ERC4626 underlying the oracle.