yAudit Timeless Bunni Oracle Review

Review Resources:

- None beyond the code.

Auditors:

- blockdev
- pandadefi

Table of Contents

Review Summary

Bunni Oracle

Timeless Bunni Oracle is a price oracle for Bunni tokens & Uniswap v3 LP positions using Chainlink & Uniswap v3 TWAP oracles.

The contracts of the Bunni Oracle Repo were reviewed over 8 days. The code review was performed by 2 auditors between July 3 and July 12, 2023. The review was limited to the latest commit at the start of the review. This was commit ca510d8ce96195d4eed732a449f04504ca6d0daf for the Bunni Oracle repo.

Scope

The scope of the review consisted of the contracts in the following directories:

- src/BunniOracle.sol

After the findings were presented to the Timeless 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, Timeless and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The contract does not appear to have any explicit access control.
Mathematics	Good	The contract includes mathematical operations for computing token prices and liquidity values. The calculations seem to be appropriately implemented.
Complexity	Good	The complexity of the contract appears to be reasonable, with functions performing specific tasks and utilizing external libraries for certain calculations.
Libraries	Good	The contract imports various external libraries that are well-tested.
Decentralization	Good	The contract is working tastelessly.
Code stability	Good	Code was stable during the review.
Documentation	Low	The code is well documented.
Monitoring	Average	The contract does not have built-in monitoring capabilities. All the functions being view functions it's not changing state variables and do not need events.

Category	Mark	Description
Testing and verification	Low	We suggest adding a differential test by implementing the price calculation logic in a language like Python and fetching price from Coingecko. This will help in identifying precision issues.

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 - Atomic Bunni token price manipulation

It is feasible to manipulate the price of a Bunni token within a single transaction.

Technical Details

Through the process of depositing into the underlying uniswap v3 position of Bunni, one can manipulate the value of the Bunni token. As a result, the price of the Bunni token will experience an immediate increase.

Allowing the price of an asset to be manipulated is at risk when the token can be borrowed.

- The token can be inflated to liquidate borrowers for profit.
- The token inflation can be used to attack the lending protocol with an attack similar to the cream hack.

It's also possible for the price to be manipulated down. In the unlikely event of a single entity owning the total supply of the Bunni token, burning the tokens then redepositing into the hub will result in a price decrease.

Impact

Medium. Token price manipulation can potentially lead to attacks.

Recommendation

It is advised to exercise caution while integrating the Bunni token and ensure that it is not made available for borrowing.

Developer Response

Acknowledged.

2. Medium - Potential Uniswap v3 price oracle manipulation

A TWAP price manipulation is possible to exploit the uniswap v3 oracle.

Technical Details

The manipulation involves controlling a certain percentage of consecutive blocks within the desired TWAP timeframe. With block producers being centralized to a small number of entities, we think it's possible to manipulate the Uniswap price oracle. An assessment of TWAP Market Risk can be found here.

BunniOracle.sol#L529-L544

Impact

Medium. The longer the TWAP window, the harder the attack becomes. However, it comes with a trade-off of higher deviation from the actual price.

Recommendation

To mitigate manipulation risks, it's important to choose TWAP timeframe based on a pool TVL. The larger the pool, the greater the funds required to manipulate it.

Developer Response

Acknowledged.

Low Findings

1. Low - Compatibility with Arbitrum

The team may deploy `BunniOracle.sol` on Arbitrum in future. The current code needs to be updated before deploying on Arbitrum because of differences between the chains.

Technical Details

- Solidity version 0.8.20 switches default target EVM version to Shanghai. This introduces a new opcode `PUSH0` which is currently not supported on Arbitrum.
- If the Arbitrum sequencer is down, the L2 rollup can only be accessed through L1 optimistic rollup contracts. If a transaction is created on an L2 rollup while the sequencer is down, the transaction would be added to a queue. Because the transaction timestamp is the time when the transaction was queued, by the time the sequencer comes back online to process the queue, the price data could be outdated.

Impact

Low.

- If solidity version is updated to 0.8.20 with Shanghai EVM version, `PUSH0` may be introduced in the bytecode making it incompatible with Arbitrum.
- If Arbitrum sequencer goes down, outdated prices may be fetched.

Recommendation

- Select the appropriate solidity or EVM version before deploying to a chain other than mainnet.
- Modify `_getPriceUSDFromFeed()` to use the Chainlink example code to check sequencer uptime. You may decide to revert or use Uniswap V3 oracle if the sequencer is down.

Developer Response

Fixed at commit 1daca0f17dc1350be8b40fcefd40347d612ce840.

yAudit Response

Reviewed. Circle has since launched native USDC on more chains including Arbitrum. This means that USDC has two versions (bridged and native). Please ensure you are using the intended address.

2. Low - Chainlink doesn't necessarily use 8 decimals

Technical Details

8 decimals is the most common precision used by Chainlink price oracle denominated in USD, but not all Chainlink oracle use 8 decimals. We fetched all the USD aggregators registered with the feed registry and found that only AMPL/USD oracle has 18 decimals. This is the aggregator returned from Chainlink registry:

```
getFeed(0xD46bA6D942050d489DBd938a2C909A5d5039A161, address(840))
```

`BunniOracle` assumes a fixed decimal precision of 8 decimals from Chainlink in BunniOracle.sol#L38:

```
uint256 internal constant PRICE_BASE = 1e8; // chainlink uses 8 decimals for prices
```

The contract doesn't return the correct values for feeds with a different precision. However, AMPL is a rebasing token so Bunni oracle will likely not be used with this oracle.

Impact

Low. The contract doesn't return correct values for feeds with a different precision.

Recommendation

If AMPL/USD oracle is not supposed to be used with this contract, future oracle integrations or deployment on other chains can still invalidate this assumption. Consider calling `decimals()` on the Chainlink aggregator to fetch the correct decimals value.

Developer Response

Fixed at 5c40aa17bad00ef694fc8d5578ce6cf3e1e2b5d4.

yAudit Response

Chainlink feed decimals are hardcoded for popular tokens. For Ethereum mainnet and Arbitrum, these values are correct, but make sure to verify the correctness before deploying on new chains. You can add such a check in the deploy script.

3. Low - `_quoteUSD()` doesn't behave as expected

Technical Details

Some assumptions of the `if` clauses in `_quoteUSD()` function are not correct (audit comments are left with `@audit` tag):

```
if (address(feed0) != address(0) && address(feed1) != address(0)) {
    // both tokens have chainlink price

    // fetch prices from chainlink
    price0USD = _getPriceUSDFromFeed(feed0, token0, chainlinkPriceMaxAgeSecs);
    price1USD = _getPriceUSDFromFeed(feed1, token1, chainlinkPriceMaxAgeSecs);
} else if (uint160(address(feed0)) | uint160(address(feed1)) == uint160(address(feed0)))
{
    // feed0 != 0, feed1 == 0                        <-- @audit feed0 == 0, feed1 == 0
also enters here.
    // token0 has chainlink price

    // fetch prices from chainlink
    price0USD = _getPriceUSDFromFeed(feed0, token0, chainlinkPriceMaxAgeSecs);

    // compute price1USD using Uniswap v3 TWAP oracle
    (int24 arithmeticMeanTick,) = OracleLibrary.consult(address(pool),
uniV3OracleSecondsAgo);
    uint256 quoteAmount =
        OracleLibrary.getQuoteAtTick(arithmeticMeanTick, token1Base.safeCastTo128(),
token1, token0); // price of token1 in token0
    price1USD = quoteAmount.mulDivDown(price0USD, token0Base);
} else if (uint160(address(feed0)) | uint160(address(feed1)) == uint160(address(feed1)))
{
    // feed0 == 0, feed1 != 0
    // token1 has chainlink price

    // fetch prices from chainlink
    price1USD = _getPriceUSDFromFeed(feed1, token1, chainlinkPriceMaxAgeSecs);
```

```
    // compute price0USD using Uniswap v3 TWAP oracle
    (int24 arithmeticMeanTick,) = OracleLibrary.consult(address(pool),
uniV3OracleSecondsAgo);
    uint256 quoteAmount =
        OracleLibrary.getQuoteAtTick(arithmeticMeanTick, token0Base.safeCastTo128(),
token0, token1); // price of token0 in token1
    price0USD = quoteAmount.mulDivDown(price1USD, token1Base);
} else {  // <-- @audit can never enter this else clause since all possibilities are
exhausted before.
    // neither token has chainlink price
    // cannot compute USD price in this case
    revert BunniOracle__NoChainlinkPriceAvailable();
}
```

Impact

Low. The UX is not the same as intended. Currently, if `address(0)` is passed for `feed0` and `feed1` and if `chainlink` is set to a valid chainlink feed registry, `_quoteUSD()` will consider the chainlink price for `feed0`. From the comments, it seems like the intention is to revert for this case.

Recommendation

Update the conditions as:

```
if (address(feed0) != address(0) && address(feed1) != address(0)) {
    ...
-} else if (uint160(address(feed0)) | uint160(address(feed1)) == uint160(address(feed0)))
{
+} else if (address(feed0) != address(0)) {
    // feed0 != 0, feed1 == 0
    ....
-} else if (uint160(address(feed0)) | uint160(address(feed1)) == uint160(address(feed1)))
{
+} else if (address(feed1) != address(0)) {
    // feed0 == 0, feed1 != 0
    ....
} else {
```

```
    // neither token has chainlink price
    // cannot compute USD price in this case
    revert BunniOracle__NoChainlinkPriceAvailable();
}
```

Developer Response

Fixed at b4ca3512aaa03e54c41d49883415c006e00b3cab.

Gas Saving Findings

1. Gas - Test contracts imported

`BunniOracle.sol` imports forge's test contracts which can be removed.

Technical Details

BunniOracle.sol#L6:

```
import "forge-std/Test.sol";
```

Impact

Gas savings.

Recommendation

Remove the import.

Developer Response

Fixed at 7fb63848b5dc6473231b98abdc2aafe0a3211e91.

Informational Findings

1. Informational - Do not declare return variables without using it

It is recommended to rename the variable to an unnamed one as it is neither assigned nor returned by name. If the optimizer is not enabled, keeping the variable as it is will also result in unnecessary gas spending.

Technical Details

BunniOracle.sol#L111 BunniOracle.sol#L170 BunniOracle.sol#L225 BunniOracle.sol#L282 BunniOracle.sol#L313 BunniOracle.sol#L370 BunniOracle.sol#L422 BunniOracle.sol#L475 BunniOracle.sol#L578 BunniOracle.sol#L614

Impact

Informational.

Recommendation

Remove the unused variables.

Developer Response

Acknowledged, won't fix. We do use the optimizer so naming the return variables helps code readability.

2. Informational - `_quoteUSD()`'s `feed0` and `feed1` may not be used to fetch price

Technical Details

`_quoteUSD()` takes `feed0` and `feed1` as the chainlink feed for the two tokens. It's not immediately clear to the users that they may not be used to fetch Chainlink price if `chainlink` is set. Additionally, the caller may pass any non-zero value to make the `_quoteUSD()` if/else checks pass, since ultimately, the feed values are not used to fetch prices.

Impact

Informational.

Recommendation

Add a NatSpec comment documenting this behavior. In case it's not expected, update the code accordingly.

Developer Response

Fixed at 089fb1ff45846b190cb152a8dc26ba2c5461eeaa.

3. Informational - `else` block unnecessary

One way to simplify the code and remove a level of nesting is by removing the `else` block and directly returning the values from the `if` block.

Technical Details

BunniOracle.sol#L584-L592 BunniOracle.sol#L645-L646 BunniOracle.sol#L665-L667

Impact

Informational.

Recommendation

Remove the `else` part of the block.

Developer Response

Acknowledged, won't fix.

Final remarks

The code review has identified several areas for improvement and potential issues in the provided code. It is recommended to address the medium-level vulnerabilities related to token price manipulation and potential oracle manipulation. Additionally, the low-level issues regarding Chainlink precision, unexpected behavior in `_quoteUSD()`, and unnecessary code imports should be resolved. Lastly, the informational recommendations regarding unused variables and code simplification can be implemented for better code clarity. Overall, addressing these recommendations will help enhance the security, reliability, and efficiency of the code.

To identify potential precision issues, we've also sent the developers a basic Python script which can be run against foundry tests.
