A CONSENSYS DILIGENCE AUDIT REPORT

Aave Balancer and Uniswap v2 Price Providers

Date	August 2020

1 Document Change Log

Version	Date	Description
1.0	2020-08-17	Initial report
1.1	2020-08-28	Updated with remediation

2 Executive Summary

This report presents the results of our assessment of **Aave’s Balancer** and **Uniswap v2 Price Providers**, both of which are an extension to the existing Aave protocol. Both price providers act as an oracle that returns the price in ETH per liquidity token.

The assessment was conducted from Aug 10 to Aug 14, 2020 as part of an ongoing engagement between Aave and ConsenSys Diligence. The objective of this collaboration is a more agile and iterative approach to smart contract security vs. the 'security last' approach currently dominating in the industry.

3 Scope

Our review focused on the Balancer Price Provider Adapter (`BalancerSharedPoolPriceProvider.sol`) and the Uniswap v2 Price Provider Adapter (`UniswapV2PriceProvider.sol`).

3.1 Objectives

We focused on the following objectives for our review:

1. Ensure the absence of known security vulnerabilities
2. Ensure the contract satisfies the critical requirements of an oracle:
    1. **Availability:** it returns a value when requested.
    2. **Integrity/Authenticity:** it returns the correct value.

4 Discussion

In the recent audit of the Aave CPM Price Provider we outlined a price manipulation vector which was mitigated by implementing means of detecting price manipulation.

Both contracts under audit implement similar means to detect manipulation by comparing the price derived from the state of the 3rd party exchange to the price provided by the Chainlink oracle. If the prices differ significantly, the Chainlink price is taken as correct, and used to derive the proper asset balance for that price.

The improved design may still have issues at times of high price volatility and/or network congestion. If the Chainlink oracle becomes stale for some reason (or an attacker intentionally causes the Chainlink update to be delayed) the liquidity pool could actually reflect the market price more accurately. In that case, the stale or manipulated Chainlink price would take precedence in determining the value of the 3rd party exchange tokens. However, the Chainlink oracle price is likely harder to manipulate than the liquidity pool reserve ratio, and under normal circumstances the error margin would be limited to the size of the price change in real world markets. A sufficiently large over-collateralization requirement should be sufficient to protect against opportunistic borrowers seeking to take under-collateralized loans during times of high volatility and chain congestion.

4.1 Balancer Price Provider

The contract is parameterized on deployment. Neither the deployer nor any other party remains in direct control of the contract. However, it is important to verify the parameterization of the contract before interacting with it, epecially as the version under audit does not validate constructor arguments which might result in a deployed but incorrectly operating price provider. The main consumer of this contract is the Aave platform.

The contract consumes ChainLink Price oracle feeds via the Aaave oracle provider, interfaces with the configured balancer pool contract and only operates on finalized shared balancer pools. This means that the operator of the balancer pool cannot add/remove or change weights for the pool tokens. These values are therefore considered constant and it is safe to fetch them when deploying the contract.

The contract exposes one external main method named `latestAnswer()` which determines the referenced Balancer Pool Token (`BPT`) value based on the asset distribution and their prices in `ETH`.

The `BPT` token value is determined in three steps:

1. For every token in the pool the corresponding balance in `ETH` (`TokenBalanceEth`) is determined using a Chainlink oracle: `BalancerPoolBalance(token) * ChainLinkTokenPrice(token)`.
2. For every token in the pool it is checked whether there is a deviation of more than +/- `MAX_DEVIATION` percent from the Chainlink price: `Deviation = SpotPrice(Token0, TokenI)/ChainlinkPrice(Token0, TokenI)`.
3. Calculate the price:
    1. **No deviation**: In order to safe gas the pool token value is calculated as `Sum(TokenBalanceEth(token) for each token)/poolToken.totalSupply`.
    2. **Deviation detected**: The token value is calculated as the geometric mean rebalanced to the ChainLink price as:

`Product(TokenBalanceEth(token)^weight for each token) * k / poolToken.totalSupply`. The result is approximated using various methods.

`poolToken.totalSupply` is the total supply of the `BAL` pool token.

The specification document `code/aave-balancer-3e8367ab/Specification Balancer Shared Pool Price Provider.pdf` (`git hash-object: b411637cc3ccf0fea7915657ce29422dec97d097`) provides a discussion about potential attacks and the proof for the alternate method of determining the token price.

4.2 Uniswap v2 Price Provider

The contract is parameterized on deployment. Neither the deployer nor any other party remains in direct control of the contract. However, it is important to verify the parameterization of the contract before interacting with it, especially as the version under audit does not validate constructor arguments which might result in a deployed but incorrectly operating price provider. The main consumer of this contract is the Aave platform.

The contract exposes one external main method named `latestAnswer()` which determines the referenced Uniswap pool token in ETH. The value is determined in three steps:

1. For each token the balance in `ETH` (`TokenBalanceEth`) is determined using a Chainlink oracle: `TokenReserve * ChainLinkTokenPrice`.
2. For the token pair it is checked whether there is a deviation of more than +/- `MAX_DEVIATION` percent from the Chainlink price: `Deviation = TokenBalanceEth(tokenA) / TokenBalanceEth(tokenB)` or `Deviation = TokenBalanceEth(tokenB) / TokenBalanceEth(tokenA)`.
3. Calculate the price:
    1. **No deviation**: In order to safe gas the value is calculated as `Sum(TokenBalanceEth(token) for each token) / pair.totalSupply`.
    2. **Deviation detected**: The token value is calculated as the geometric mean rebalanced to the ChainLink price as: `2 * sqrt(TokenBalanceEth(tokenA) * TokenBalanceEth(tokenB)) / pair.totalSupply`

The `pair.totalSupply` is the Uniswap V2 pair total supply.

The specification document `code/aave-uniswapv2-e81cf872/Final Specification Uniswap V2 Price Provider.pdf` (`git hash-object: c98c8c325cfecf99a1abcfc291457ea8e2080686`) discusses potential attacks and provides a proof for the alternative method of calculating the token price. We would like to note that the audit team suggested the following minor changes to the document:

- Page 6: Ratio `R` should be named `Rio` or `Roi` as usually `Rio != Roi` with `Rio = 1/Roi`
- Page 6: The formula for the first 1st and the 2nd ratio do not match. the first ratio is `Rio` and the 2nd is `Roi` which might be confusing
- Page 8: eq2 `SPyx = y/x` should be `SPyx = x/y` , hence eq4 should be `y/x = Py/Px` , therefore, eq10 cannot be simplified to eq11.

The issues were partially addressed with gitlab revision: b3c66a57, fixing the error on Page 8. However, eq2 should be `SPyx = x / y` and eq3 should be `CPyx = Px/Py` according to the definitions on page 6. This, however, does not change the resulting simplified formula for the geometric mean.

5 Recommendations

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Code readability - Rename `priceDeviation` to `maxPriceDeviation` ✓ Fixed

Resolution

The variable was renamed.

Description

Improve code readability by renaming the state variable `priceDeviation` to `maxPriceDeviation`, distinguishing it from the local variable `price_deviation` and indicating that the variable is a limit as outlined in the specification (`MAX_DEVIATION`).

Balancer

code/aave-balancer-3e8367ab/contracts/proxies/BalancerSharedPoolPriceProvider.sol:L124-L129

```
if (
    price_deviation > (BONE + priceDeviation) ||
    price_deviation < (BONE - priceDeviation)
) {
    return true;
}
```

Uniswapv2

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L83-L95

```
if (
    price_deviation > (Math.BONE + priceDeviation) ||
    price_deviation < (Math.BONE - priceDeviation)
) {
    return true;
}
price_deviation = Math.bdiv(ethTotal_1, ethTotal_0);
if (
    price_deviation > (Math.BONE + priceDeviation) ||
    price_deviation < (Math.BONE - priceDeviation)
) {
    return true;
}
```

5.2 Improve Input Validation ✓ Fixed

Resolution

the recommended checks have been added to the constructor.

Description

The constructor does not validate whether the provided price provider arguments actually make sense. In the worst-case someone might be able to deploy the contract that cannot be used. It is recommended to fail the contract creation early if invalid arguments are detected.

Consider implementing the following checks to detect whether a non-viable price provider is being deployed:

- `tokens.length > 1` and less than the maximum supported tokens (note that `hasDeviation` requires `token.length**2` iterations if no deviation is detected)
- `_isPeggedToEth.length == tokens.length`
- `_decimals.length == tokens.length`
- `approximationMatrix.length && approximationMatrix[0][0].length == tokens.length +1`
- `_priceDeviation` is within bounds (less than 100%, i.e. less than `1 * BONE`) otherwise the calculation might underflow.
- `_powerPrecision` is within bounds
- `address(_priceOracle) != address(0)`

Balancer

code/aave-balancer-3e8367ab/contracts/proxies/BalancerSharedPoolPriceProvider.sol:L38-L63

```
constructor(
    BPool _pool,
    bool[] memory _isPeggedToEth,
    uint8[] memory _decimals,
    IPriceOracle _priceOracle,
    uint256 _priceDeviation,
    uint256 _K,
    uint256 _powerPrecision,
    uint256[][] memory _approximationMatrix
) public {
    pool = _pool;
    //Get token list
    tokens = pool.getFinalTokens(); //This already checks for pool finalized
    //Get token normalized weights
    uint256 length = tokens.length;
    for (uint8 i = 0; i < length; i++) {
        weights.push(pool.getNormalizedWeight(tokens[i]));
    }
    isPeggedToEth = _isPeggedToEth;
    decimals = _decimals;
    priceOracle = _priceOracle;
    priceDeviation = _priceDeviation;
    K = _K;
    powerPrecision = _powerPrecision;
    approximationMatrix = _approximationMatrix;
}
```

Uniswapv2

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L35-L50

```
constructor(
    IUniswapV2Pair _pair,
    bool[] memory _isPeggedToEth,
    uint8[] memory _decimals,
    IPriceOracle _priceOracle,
    uint256 _priceDeviation
) public {
    pair = _pair;
    //Get tokens
    tokens.push(pair.token0());
    tokens.push(pair.token1());
    isPeggedToEth = _isPeggedToEth;
    decimals = _decimals;
    priceOracle = _priceOracle;
    priceDeviation = _priceDeviation;
}
```

5.3 Use SafeMath consistently ✓ Fixed

Resolution

All arithmetic operations now use SafeMath.

Description

Even though the Uniswap price provider imports the `SafeMath` library, the SafeMath library functions aren't always used for integer arithmetic operations. Note that plain Solidity arithmetic operators do not check for integer underflows and overflows.

Examples

Example 1:

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L66

```
uint256 missingDecimals = 18 - decimals[index];
```

Example 2 (same in line 91-92):

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L84-L85

```
price_deviation > (Math.BONE + priceDeviation) ||
price_deviation < (Math.BONE - priceDeviation)
```

Example 3:

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L164-L165

```
uint256 liquidity = numerator / denominator;
totalSupply += liquidity;
```

Recommendation

In some cases, this issue is cosmetic because the values are assumed to be within certain ranges. Nevertheless, we recommend accepting the slightly higher gas cost for SafeMath functions for consistency and to prevent potential issues.

6 Issues

The issues are presented in approximate order of priority from highest to lowest.

6.1 Unchecked Specification requirement - token limit

Closed

Description

According to the `Balancer Shared Pool Price Provider` that was provided with the audit code-base the price provide must fulfill the following requirements:

- Pool token price cannot be manipulated
- Chainlink will be used as the main oracle
- It should use as less gas as possible
- Limited to Balancer's shared pools where the weights cannot be changed
- Limited to a pool containing 2 to 3 tokens

However, the constructor of the price provider does not enforce the limit of 2 to 3 tokens.

Examples

code/aave-balancer-3e8367ab/contracts/proxies/BalancerSharedPoolPriceProvider.sol:L38-L63

```
constructor(
    BPool _pool,
    bool[] memory _isPeggedToEth,
    uint8[] memory _decimals,
    IPriceOracle _priceOracle,
    uint256 _priceDeviation,
    uint256 _K,
    uint256 _powerPrecision,
    uint256[][] memory _approximationMatrix
) public {
    pool = _pool;
    //Get token list
    tokens = pool.getFinalTokens(); //This already checks for pool finalized
    //Get token normalized weights
    uint256 length = tokens.length;
    for (uint8 i = 0; i < length; i++) {
        weights.push(pool.getNormalizedWeight(tokens[i]));
    }
    isPeggedToEth = _isPeggedToEth;
    decimals = _decimals;
    priceOracle = _priceOracle;
    priceDeviation = _priceDeviation;
    K = _K;
    powerPrecision = _powerPrecision;
    approximationMatrix = _approximationMatrix;
}
```

Recommendation

Require that the number of tokens returned by `pool.getFinalTokens()` is `2<= len <=3`.

6.2 Integer underflow if a token specifies more than 18 decimals Closed

Description

Decimals are provided by the account deploying the price provider contract. In `getEthBalanceByToken` the assumption is made that `decimals[index]` is less or equal to `18` decimals, however, the deployer may provide decimals that are not within normal operating bounds. Contract creation succeeds, while the contract is not viable.

Examples

The value underflows if the contract is used with a token decimals > 18.

Balancer

code/aave-balancer-3e8367ab/contracts/proxies/BalancerSharedPoolPriceProvider.sol:L69-L78

```
function getEthBalanceByToken(uint256 index)
    internal
    view
    returns (uint256)
{
    uint256 pi = isPeggedToEth[index]
        ? BONE
        : uint256(priceOracle.getAssetPrice(tokens[index]));
    require(pi > 0, "ERR_NO_ORACLE_PRICE");
    uint256 missingDecimals = 18 - decimals[index];
```

Uniswapv2

code/aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol:L57-L66

```
function getEthBalanceByToken(uint256 index, uint112 reserve)
    internal
    view
    returns (uint256)
{
    uint256 pi = isPeggedToEth[index]
        ? Math.BONE
        : uint256(priceOracle.getAssetPrice(tokens[index]));
    require(pi > 0, "ERR_NO_ORACLE_PRICE");
    uint256 missingDecimals = 18 - decimals[index];
```

Recommendation

Add a check to the constructor to ensure that none of the provided decimals is greater than 18.

Appendix 1 - Files in Scope

This audit covered the following files:

4.1 Balancer Price Provider

Revision of the repository under audit: `3e8367ab211a137afff87dd8dadc0efe235257d4`

File Name	SHA-1 Hash	git hash-object
aave-balancer-3e8367ab/contracts/proxies/BalancerSharedPoolPriceProvider.sol	d13214588b26fd856f06b51873113acbc59b7950	f4cfa7bcfaf856a12b293d5e13bcb58911c0a4f6

Out of Scope

Files Excluded for being an Interface or a copy of an audited 3rd party component.

File Name	SHA-1 Hash	git hash-object
aave-balancer-3e8367ab/contracts/interfaces/IPriceOracle.sol	35cf8e4c5cd0035e44484a3b8202f65d06f990b8	ac83f98fed040b3fcf555f858327fef9bd045fd4
aave-balancer-3e8367ab/contracts/interfaces/BPool.sol	6f49423eb769a025081b448ea96acac30958ba5f	005437d8c9cfc428bbac032f93c86e285924f54b
aave-balancer-3e8367ab/contracts/misc/BConst.sol	beeccc9a3f683651c146507f1c1aeba4b4db12f2	48bc5cd9493d44d8151c8dcc13029cf976377000
aave-balancer-3e8367ab/contracts/misc/BNum.sol	4ae208b6caa2e45491e86063349c5e9cf8c47b9d	58c3824c8913aeae73da5b898e10c7bf43db925b

- `BConst.sol` was verified to be an unmodified copy of balancer-labs/Bconst.sol
- `BNum.sol` was verified to be an unmodified copy of balancer-labs/BNum.sol

4.2 Uniswap v2 Price Provider

Revision of the repository under audit: `e81cf872e3c08d7c43c1e1d2e90dffa01844230e`

File Name	SHA-1 Hash	git hash-object
aave-uniswapv2-e81cf872/contracts/proxies/UniswapV2PriceProvider.sol	aa267c067d7be83642a9555cd190a0cf136d9bea	e2702ae70d7804d882b8996a13bb2b7ab6af9693

Out of Scope

Files Excluded for being an Interface or a copy of an audited 3rd party component.

File Name	SHA-1 Hash	git hash-object
aave-uniswapv2-e81cf872/contracts/misc/SafeMath.sol	f9900f7586bbc0c1362e1b1c93d5054c8a0ae277	38921c90eb539aaea011cf1b75a54f69d2466982
aave-uniswapv2-e81cf872/contracts/misc/Math.sol	1dfd0f12e6967a547d3161d2fbf75105147d920b	9bc3570b330a72ce1973ec94b329d4a1ce22e555
aave-uniswapv2-e81cf872/contracts/interfaces/IPriceOracle.sol	35cf8e4c5cd0035e44484a3b8202f65d06f990b8	ac83f98fed040b3fcf555f858327fef9bd045fd4
aave-uniswapv2-e81cf872/contracts/interfaces/IUniswapV2Factory.sol	8fe4e07b64e4820bbda4386c9c903c868de4d23f	0a9cb0b57533c3b026d982ab830c63f786ba7c27
aave-uniswapv2-e81cf872/contracts/interfaces/IUniswapV2Pair.sol	bef5664cddbb670e01ad20ea59e66e66c2b1e02d	3b40ec05a7a18ba24b62a81908311e10fcf85e3a

- `SafeMath.sol` was verified to be an unmodified copy of dsmath/math.sol
- `Math.sol` includes methods where all but `bsqrt` are identical to balancer-labs/BNum.sol

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.