ABDK CONSULTING

SMART CONTRACT AUDIT

Maker

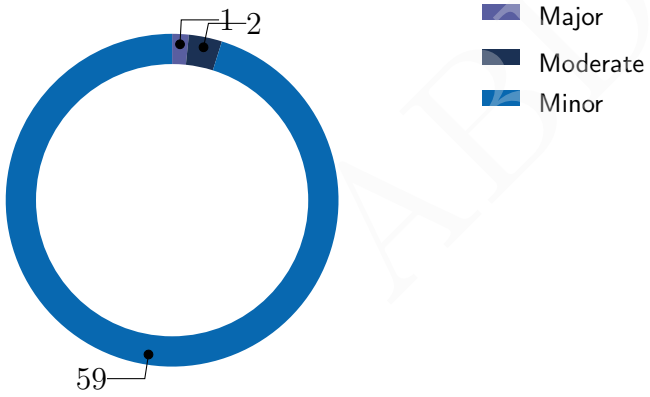
Univ2LpOracle

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
15th March 2021

We've been asked to review the smart contracts given in separate files. At some point we were also given the formal spec.



Findings

ID	Severity	Subject	Status
CVF-1	Minor	Inconsistent notation.	Opened
CVF-2	Minor	Improper comment	Opened
CVF-3	Minor	Complicated code	Opened
CVF-4	Minor	Improper Solidity version	Opened
CVF-5	Minor	Improper interface placement	Opened
CVF-6	Minor	ERC20 inconsistency	Opened
CVF-7	Minor	Unclear semantic	Opened
CVF-8	Minor	Improper interface placement	Opened
CVF-9	Minor	Bad naming	Opened
CVF-10	Minor	Bad naming	Opened
CVF-11	Minor	Bad naming	Opened
CVF-12	Minor	Unspecific types	Opened
CVF-13	Minor	Improper approach	Opened
CVF-14	Minor	Unspecific types	Opened
CVF-15	Minor	Bad naming	Opened
CVF-16	Minor	Improper codestyle	Opened
CVF-17	Minor	Unclear semantic	Opened
CVF-18	Minor	Bad naming	Opened
CVF-19	Minor	Redundant code	Opened
CVF-20	Minor	Redundant code	Opened
CVF-21	Minor	Improper type	Opened
CVF-22	Minor	Standard inconsistency	Opened
CVF-23	Minor	Bad naming	Opened
CVF-24	Minor	Comment missing	Opened
CVF-25	Minor	Unclear type meaning	Opened
CVF-26	Minor	Redundant code	Opened
CVF-27	Minor	Improper type	Opened

ID	Severity	Subject	Status
CVF-28	Minor	Bad naming	Opened
CVF-29	Minor	Redundant function	Opened
CVF-30	Minor	Overflow	Opened
CVF-31	Minor	Overflow	Opened
CVF-32	Minor	Improper implementation	Opened
CVF-33	Minor	Return missing	Opened
CVF-34	Minor	Improper type	Opened
CVF-35	Minor	Improper type	Opened
CVF-36	Minor	Unspecific type	Opened
CVF-37	Minor	Complicated code	Opened
CVF-38	Minor	Undefined variable	Opened
CVF-39	Minor	Incompatible code	Opened
CVF-40	Minor	Redundant code	Opened
CVF-41	Minor	Redundant code	Opened
CVF-42	Minor	Improper type	Opened
CVF-43	Major	Check missing	Opened
CVF-44	Minor	Redundant code	Opened
CVF-45	Minor	Improper value	Opened
CVF-46	Minor	Redundant code	Opened
CVF-47	Minor	Improper type	Opened
CVF-48	Minor	Condition missing	Opened
CVF-49	Minor	Redundant code	Opened
CVF-50	Minor	Expensive code	Opened
CVF-51	Minor	Improper approach	Opened
CVF-52	Minor	Improper approach	Opened
CVF-53	Minor	Complicated code	Opened
CVF-54	Minor	Gas spending	Opened
CVF-55	Minor	Confusing comment	Opened
CVF-56	Moderate	Overflow	Opened
CVF-57	Minor	Redundant code	Opened

ID	Severity	Subject	Status
CVF-58	Moderate	Overflow	Opened
CVF-59	Minor	Precision degradation	Opened
CVF-60	Minor	Overflow	Opened
CVF-61	Minor	Unclear parameter meaning	Opened
CVF-62	Minor	Redundant code	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Mar. 13, 2021	D. Khovratovich	Initial Draft
0.2	Mar. 14, 2021	D. Khovratovich	Minor Revision
1.0	Mar. 15, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1 Inconsistent notation.

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description The two underlying tokens are referred as 0 and 1 here, but as X and Y below.
Recommendation Consider using consistent notation.

Listing 1: Inconsistent notation.

```
26 INVARIANT k = reserve0 [num token0] * reserve1 [num token1]
```

3.2 CVF-2 Improper comment

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Univ2LpOracle.sol

Description Prices p_x and p_y here are the prices offered by the Uniswap pool, not necessary the current market prices or fair prices. Also, it is unclear what terms prices p_x and p_y are denominated in. A Uniswap pool doesn't deal with the prices of the underlying tokens, but only with their mutual exchange rate.

Listing 2: Improper comment

```
32 r_x * p_x = r_y * p_y    // Proportion of r_x and r_y can be
   ↪ manipulated so need to normalize them
```

3.3 CVF-3 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation This formula could be rewritten as: 2 *gavg (v_x, v_y) / total_supply, where v_x and v_y are the market values of the underlying token reserves in the pool.

Listing 3: Complicated code

```
41 p_lp = (r_x * p_x + r_y * p_y) / supply_lp = 2 * sqrt(k * p_x *
   ↪ p_y) / supply_lp
```

3.4 CVF-4 Improper Solidity version

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation Should be "̂0.6.0" according to a common best practice.

Listing 4: Improper Solidity version

```
43 solidity ^0.6.11;
```

3.5 CVF-5 Improper interface placement

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation These interfaces should probable be defined in their own files.

Listing 5: Improper interface placement

```
45 ERC20Like {

51 UniswapV2PairLike {

58 OracleLike {
```

3.6 CVF-6 ERC20 inconsistency

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The return types should be ERC20Like.

Listing 6: ERC20 inconsistency

```
53 function token0()      external view returns (address);
   function token1()      external view returns (address);
```

3.7 CVF-7 Unclear semantic

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Univ2LpOracle.sol

Description The semantic of the second (bool) returned value is unclear.
Recommendation Consider giving it a descriptive name and/or adding a documentation comment.

Listing 7: Unclear semantic

```
60 function peek() external view returns (uint256,bool);
```

3.8 CVF-8 Improper interface placement

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Univ2LpOracle.sol

Recommendation This contract should be defined in its own file.

Listing 8: Improper interface placement

```
64 UNIV2LPOracleFactory {
```

3.9 CVF-9 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Recommendation Events are usually named via nouns, such as "Creation" or "NewOracle".

Listing 9: Bad naming

```
68 event Created(address sender, address orcl, bytes32 wat, address
   ↪ tok0, address tok1, address orb0, address orb1);
```

3.10 CVF-10 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description Abbreviated names of event parameters make code less readable. Also, remember, that unlike names of function parameters, the names of event parameters are a part of smart contract public API.
Recommendation Consider using non-abbreviated parameter names.

Listing 10: Bad naming

```
68 event Created(address sender, address orcl, bytes32 wat, address
   ↪ tok0, address tok1, address orb0, address orb1);
```

3.11 CVF-11 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description It is unclear that is wat parameter here.
Recommendation Consider giving this parameter more descriptive name.

Listing 11: Bad naming

```
68 event Created(address sender, address orcl, bytes32 wat, address
   ↪ tok0, address tok1, address orb0, address orb1);
```

3.12 CVF-12 Unspecific types

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation Consider using more specific types for event parameters, such as UNIV2LPOracle for orc1, OracleLike for orb0 and orb1, ERC20Like for tok0 and tok1.

Listing 12: Unspecific types

```
68 event Created(address sender, address orcl, bytes32 wat, address
   ↪ tok0, address tok1, address orb0, address orb1);
```

3.13 CVF-13 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation Consider declaring some event parameters as indexed, for example orc1, tok1, and tok2.

Listing 13: Improper approach

```
68 event Created(address sender, address orcl, bytes32 wat, address
    ↪ tok0, address tok1, address orb0, address orb1);
```

3.14 CVF-14 Unspecific types

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation Consider using more specific types for function parameters, such as UniswapV2PairLike for _src and OracleLIke for _orb0 and _orb1. Also consider using more specific type for the returned value, such as UNIV2LPOracle.

Listing 14: Unspecific types

```
71 function build(address _src, bytes32 _wat, address _orb0,
    ↪ address _orb1) public returns (address orcl) {
```

3.15 CVF-15 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description The contract is named UNIV2LPOracle, while the file is named Univ2LpOracle.
Recommendation Consider using consistent letter cases.

Listing 15: Bad naming

```
81 UNIV2LPOracle {
```

3.16 CVF-16 Improper codestyle

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Univ2LpOracle.sol

Description In the body of this contract storage variables, modifiers and functions are intermixed.
Recommendation Consider grouping them somehow to make code more readable.

Listing 16: Improper codestyle

```
81 UNIV2LPOracle {
```

3.17 CVF-17 Unclear semantic

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Univ2LpOracle.sol

Description The semantics of the values in this mapping is unclear.
Recommendation Consider adding more details into the documentation comment above.

Listing 17: Unclear semantic

```
84 mapping (address => uint) public wards;
   ↪                                      // Addresses with
   ↪ admin authority
```

3.18 CVF-18 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description Whether or not to use underscore ('_') prefix for function parameters depends mostly on personal preference, but naming policy should be consistent across the code to make the code more readable. Currently, parameters of some functions are prefixed with underscore, while parameters of some other functions are not prefixed.

Listing 18: Bad naming

```
85 function rely(address usr) external auth { wards[usr] = 1; emit
   ↪ Rely(usr); }  // Add admin
```

3.19 CVF-19 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description Here "Rely" event is logged even if the user was already an admin.

Listing 19: Redundant code

```
85 function rely(address usr) external auth { wards[usr] = 1; emit
   ↪ Rely(usr); }  // Add admin
```

3.20 CVF-20 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description Here "Deny event is logged even if the user wasn't an admin.

Listing 20: Redundant code

```
86 function deny(address usr) external auth { wards[usr] = 0; emit
   ↪ Deny(usr); }  // Remove admin
```

3.21 CVF-21 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Description The type should be UniswapV2PairLike.

Listing 21: Improper type

```
92 address public src;                  // Price source
```

3.22 CVF-22 Standard inconsistency

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Description The type uint16 is actually less efficient in Solidity than uint256, why is it used here?
Recommendation Consider using uint256 for time stamps and time intervals, as this is a standard best practice.

Listing 22: Standart inconsistency

```
93 uint16 public hop = 1 hours; // Minimum time inbetween price
   ↪ updates
```

3.23 CVF-23 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Description The name of this storage variable is odd. Remember, that names of public storage variables are a part of public API of smart contract.

Listing 23: Bad naming

```
94 uint64 public zzz; // Time of last price update
```

3.24 CVF-24 Comment missing

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Description Why is it bytes32? It this a hash of something?
Recommendation Consider adding more details into the comment.

Listing 24: Comment missing

```
95 bytes32 public immutable wat; // Token whose price is being
   ↪ tracked
```

3.25 CVF-25 Unclear type meaning

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Univ2LpOracle.sol

Description If this is a boolean flag, why does it have type uint128?

Listing 25: Unclear type meaning

```
103 uint128 has; // Is price valid
```

3.26 CVF-26 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation The contract actually uses the product of these normalizers, so storing them separately is redundant.

Listing 26: Redundant code

```
114 uint256 private immutable normalizer0; // Multiplicative factor
    ↪ that normalizes a token0 balance to a WAD; 10^(18 - dec)
uint256 private immutable normalizer1; // Multiplicative factor
    ↪ that normalizes a token1 balance to a WAD; 10^(18 - dec)
```

3.27 CVF-27 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The types should be OracleLike.

Listing 27: Improper type

```
117 address public orb0; // Oracle for token0, ideally a
    ↪ Medianizer
address public orb1; // Oracle for token1, ideally a
    ↪ Medianizer
```

3.28 CVF-28 Bad naming

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Univ2LpOracle.sol

Recommendation 1e18 would be shorter.

Listing 28: Bad naming

```
121 uint256 constant WAD = 10 ** 18;
```

3.29 CVF-29 Redundant function

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Description This function is not used and should be removed. Also, it looks confusing, as it requires x to be a factor y.
Recommendation Consider removing this function.

Listing 29: Redundant function

```
132 function div(uint x, uint y) internal pure returns (uint z) {
        require(y > 0 && (z = x / y) * y == x, "ds-math-divide-by-
            ↪ zero");
    }
```

3.30 CVF-30 Overflow

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Univ2LpOracle.sol

Description Phantom overflow is possible here, i.e. a situation when the final result would fit into the destination type, but some intermediary value overflow.
Recommendation Consider using simple math tricks described here: https://medium.com/coinmonks/math-in-solidity-part-3-percents-and-proportions-4db014e080b1

Listing 30: Overflow

```
136 z = add(mul(x, y), WAD / 2) / WAD;
```

3.31 CVF-31 Overflow

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Univ2LpOracle.sol

Description Phantom overflow is possible here.

Listing 31: Overflow

```
139 z = add(mul(x, WAD), y / 2) / y;
```

3.32 CVF-32 Improper implementation

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Description The implementation of this function is suboptimal.
Recommendation Look for example to how this is implemented in the ABDKMath64x64 library: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/ABDKMath64x64.sol#L687-L709

Listing 32: Improper implementation

```
142 function sqrt(uint y) internal pure returns (uint z) {
```

3.33 CVF-33 Return missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Univ2LpOracle.sol

Description Not all branches of the function return a value.

Listing 33: Return missing

```
142 function sqrt(uint y) internal pure returns (uint z) {
```

3.34 CVF-34 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The parameter type should be UniswapV2PairLike.

Listing 34: Improper type

```
158 event Change(address indexed src);
```

3.35 CVF-35 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The type of the second parameter should be OracleLike.

Listing 35: Improper type

```
163 event Link(uint256 id, address orb);
```

3.36 CVF-36 Unspecific type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation Consider using more specific types for the parameters such as UniswapV2PairLike for _src and OracleLike for _orb0 and _orb1.

Listing 36: Uspecific type

```
168 constructor(address _src, bytes32 _wat, address _orb0, address
    ↪ _orb1) public {
```

3.37 CVF-37 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation Too separate require statements would make problems easier to investigate.

Listing 37: Complicated code

```
170 require(_orb0 != address(0) && _orb1 != address(0), "
    ↪ UNIV2LPOracle/invalid-oracle-address");
```

3.38 CVF-38 Undefined variable

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation This could be done at the definition of "zzz" variable.

Listing 38: Undefined variable

```
173 zzz = 0;
```

3.39 CVF-39 Incompatible code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This code is not compatible with tokens that have more than 18 decimals.
Recommendation Consider making the normalizers to be WADs to allow up to 36 decimals.

Listing 39: Incompatible code

```
175 normalizer0 = 10 ** sub(18, uint256(ERC20Like(UniswapV2PairLike(
    ↪ _src).token0()).decimals())); // Calculate normalization
    ↪ factor of token0
    normalizer1 = 10 ** sub(18, uint256(ERC20Like(UniswapV2PairLike(
    ↪ _src).token1()).decimals())); // Calculate normalization
    ↪ factor of token1
```

3.40 CVF-40 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This event is logged even when the contract was already stopped.

Listing 40: Redundant code

```
186 emit Stop();
```

3.41 CVF-41 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This event is logged even when the contract wasn't stopped.

Listing 41: Redundant code

```
191 emit Start();
```

3.42 CVF-42 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The parameter type should be UniswapV2PairLike.

Listing 42: Improper type

```
194 function change(address _src) external auth {
```

3.43 CVF-43 Check missing

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This allows setting a source to be a Uniswap pair whose tokens differ from the original tokens, thus making current oracles irrelevant. Even worse, the new tokens may have different numbers of decimals making normalizers irrelevant.
Recommendation Consider ensuring that tokens are the same. Alternatively, consider atomically updating the source together with the oracles and the normalizers to ensure consistency.

Listing 43: Check missing

```
195 src = _src;
```

3.44 CVF-44 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Description This event is logged even when the new source is the same is the current one.

Listing 44: Redundant code

```
196 emit Change(src);
```

3.45 CVF-45 Improper value

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Univ2LpOracle.sol

Description Using uint16(-1) seconds, i.e. 18 hours, 12 minutes, and 15 seconds as the maximum allowed hop looks quite strange.
Recommendation Consider using more reasonable value, such as 24 hours, and making this value a named constant.

Listing 45: Improper value

```
200 require(_hop <= uint16(-1), "UNIV2LPOracle/invalid-hop");
```

3.46 CVF-46 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This event is logged even when the new hop is the same as the current one.

Listing 46: Redundant code

```
202 emit Step(hop);
```

3.47 CVF-47 Improper type

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Univ2LpOracle.sol

Recommendation The type of the "orb" parameter should be OracleLIke.

Listing 47: Improper type

```
205 function link(uint256 id, address orb) external auth {
```

3.48 CVF-48 Condition missing

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Univ2LpOracle.sol

Description There is no range check for the "id" parameter, so when id is invalid, the function doesn't change contract's state but still logs an event.
Recommendation Consider adding "else revert();" here.

Listing 48: Condition missing

```
211 }
```

3.49 CVF-49 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This event is logged even if the new oracle is the same as the current one, or when "id" is invalid.

Listing 49: Redundant code

```
212 emit Link(id, orb);
```

3.50 CVF-50 Expensive code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation This code would be cheaper in case zzz will be stored with hop already added to it. Then the correction of hop, if it is rare, can be done together with the update of zzz.

Listing 50: Expensive code

```
216 return block.timestamp >= add(zzz, hop);
```

3.51 CVF-51 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This function performs calculations in an overflow-prone and precision degradation-prone way. It also doesn't support token with more than 18 decimals. More reliable and precise way would be to first calculate the market values of token reserves by multiplying the normalized balances by the corresponding oracle prices. As long as market values are all in USD, their absolute values cannot be too high and their reasonable precision is known, so the y could be represented as 128-bit WADs. Then geometric mean of the calculated marked values could be calculated, doubled and divided by the total supply of LP tokens. This way, the function will work even for extremely cheap tokens with enormous total supply. The only limitation would be that the market value value of the token reserves should not exceed $340,282,370T which looks quite safe assumption.

Listing 51: Improper approach

```
219 function seek() internal returns (uint128 quote, uint32 ts) {
```

3.52 CVF-52 Improper approach

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation You may use "ts" here, no need for a seaprate local variable.

Listing 52: Improper approach

```
224 (uint112 res0, uint112 res1, uint32 _ts) = UniswapV2PairLike(src
    ↪ ).getReserves();
```

3.53 CVF-53 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Recommendation Two separate require statements would make problems easier to investigate.

Listing 53: Complicated code

```
225 require(res0 > 0 && res1 > 0, "UNIV2LPOracle/invalid-reserves");
```

3.54 CVF-54 Gas spending

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Description What is the reason to return "ts" from the function once it always equals to the current block timestamp? Looks like waste of gas.

Listing 54: Gas spending

```
227 require(ts == block.timestamp);
```

3.55 CVF-55 Confusing comment

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Univ2LpOracle.sol

Description This comment is confusing. There is no good reason why overflow is not possible, and the most of the code does all math in an overflow-protected way, even when overflow is very improbably. Consider just using safe math consistently through the code.

Listing 55: Confusing comment

```
230 // TODO: is the risk of overflow here worth mitigating? (
    ↪ consider an attacker who can mint a token at will)
```

3.56 CVF-56 Overflow

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Univ2LpOracle.sol

Description Overflow is possible here.

Listing 56: Overflow

```
231 if (normalizer1 > 1) res1 = uint112(res1 * normalizer1);
```

3.57 CVF-57 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Univ2LpOracle.sol

Description Conditional statement costs more than a multiplication, so these conditional statement don't save any gas.

Listing 57: Redundant code

```
231 if (normalizer1 > 1) res1 = uint112(res1 * normalizer1);
```

3.58 CVF-58 Overflow

- **Severity** Moderate
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Univ2LpOracle.sol

Description Overflow is possible during conversion to uint128.

Listing 58: Overflow

```
247 quote = uint128(
```

3.59 CVF-59 Precision degradation

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description Precision degradation is possible here, as wmul and sqrt calls perform roundings in the middle of the calculations.

Listing 59: Precision degradation

```
248 mul(2 * WAD, sqrt(wmul(k, wmul(val0, val1))))
```

3.60 CVF-60 Overflow

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Univ2LpOracle.sol

Description Phantom overflow is possible here.

Listing 60: Overflow

```
248 mul(2 * WAD, sqrt(wmul(k, wmul(val0, val1))))
```

3.61 CVF-61 Unclear parameter meaning

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Univ2LpOracle.sol

Description The meanings of the second parameters are unclear.
Recommendation Consider giving them descriptive names and/or adding documentation comments.

Listing 61: Uclear paremeter meaning

```
263 function peek() external view toll returns (bytes32,bool) {

267 function peep() external view toll returns (bytes32,bool) {
```

3.62 CVF-62 Redundant code

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Univ2LpOracle.sol

Description This event is logged even in case the status of the address didn't actually change.

Listing 62: Redundant code

```
279 emit Kiss(a);

286     emit Kiss(a[i]);

292 emit Diss(a);

298     emit Diss(a[i]);
```