Smart Contract Audit

One Oracle

coinspect

One Oracle

Smart Contract Audit

V220517

Prepared for Vesper • May 2022

1. Executive Summary

In **May 2022**, **Vesper** engaged Coinspect to perform a source code review of **One Oracle**. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
0	4	2
Fixed 0	Fixed 0	Fixed 0

2. Assessment and Scope

The audit started on **May 10** and was conducted on the **main** branch of the git repository at bloqpriv/one-oracle as of commit **`e9ee30f3a0aac2e241397b4d0e772d8497848aef`** of **May 12**.

The audited files have the following sha256sum hash:

```
2974c9355308f0fa3c7f51fa466251b377aafe5f7e9e43b7e2f770a764d25f83  swapper/Swapper.sol
d43842f1393df6d7c29720683bf6f48a03fa324f55cbd35bbe9b6eaaa12341ee  swapper/UniswapV2LikeExchange.sol
008d3c9950fcdbd55a1ef851712c241fc6e1ce8fa8da86fbfe1bb9120a6f433d  libraries/DataTypes.sol
6d86573351b7d925665b34e5f675495059b9d14dcb94e8d131a30b38d66e0234  libraries/SafeUint128.sol
a1708fd0bc50f215a589782983869e6ebff8abfc5187ee7c16eb9ca765300748  libraries/OracleHelpers.sol
ae13849267cbba43e3b024ffe0feaa52fef92c2830cbfc6c0cafab0e52434292  periphery/Oracle.sol
b85f64f0dcd82ddec8423b3d13686a3a384e3597ef9c6813821f824f311cb361  periphery/SwapperOracle.sol
06aa5bd6e622b2427342784855cc73491b2c89d426a014bcd277b7d759a74dd0  periphery/synth/SynthDefaultOracle.sol
a219f60d182f33358a6a8e17fd83943e6ee2e13bca0f28b2b047c39f2dd59018  periphery/synth/CurveLpTokenOracle.sol
d39396d31c38860333cfb03b2517467bd5f971e0b79a0432ffe6e3328b05f8b9  periphery/synth/CTokenOracle.sol
8dfb1fe8dd4e15b744112d7eddf64cd869c70787e37d86eb8b9ae83815c99e4b  interfaces/IGovernable.sol
636438b8a30611c5eb3854cab806d44b0c5a3a8b2623446167f6cc742cd8b1fd  interfaces/swapper/ISwapper.sol
1909dd84d58fe700004203c760baf21afe60bf5f45cb8c1ab1a2f5f4c0147935  interfaces/swapper/IExchange.sol
b2ba9faafa3fbc2b4c1fc37e457906dd53a56fe1b77b704746b4f2439b24b1f6  interfaces/external/ICToken.sol
5b11f5d97f6e766f572bf1a4e611f597ce380ff358aa64ecfbc64766f90f8e06  interfaces/external/ICurveAddressProvider.sol
fa8285d12e197c9ae063b4209408309d7bcd2cafe91f32e76d4ed951a20abeff  interfaces/external/ICurveRegistry.sol
001855feb87f6bec9329ffd1624a6d805923a402f6c3ca110d06225bc0508c47  interfaces/periphery/IOracle.sol
e4d513da2b41abd5e94370012b2a14815ebae02d49fbf510114d5648dea69fc0  interfaces/periphery/ISwapperOracle.sol
8aaf922d29b0d2735f9b102a2334529d4e7e2de993e94df008ebe18f01846adb  interfaces/periphery/synth/ISynthOracle.sol
c6f4b53d5ee9ca6b73eeeb729ca550ec1c0dd01b480b3da714d40a5a8a02f08b  interfaces/core/IUniswapV2LikePriceProvider.sol
c3f13a33b8f5f8ee6856f396235785004bb1419768ff9c019551771e2818c73b  interfaces/core/IPriceProvidersAggregator.sol
c282023a69f6b9633d60ebd7bbffd1099cfd4df8938b84357b3e21cd5699d4ae  interfaces/core/IUniswapV3PriceProvider.sol
48b3a6b429bb5c82429bd617cfa9e15d3eca85114f01fb9fea474ac4f554fb94  interfaces/core/IChainlinkPriceProvider.sol
8923f2f1c41f4b3b20c95dac66b9960744e420e09068b118750b069d06893a75  interfaces/core/IUSDPriceProvider.sol
5c1256282a80d687f10fb3df07bbad3dbc10f0b90976c5c7b3a2862aa7714ba2  interfaces/core/IUmbrellaPriceProvider.sol
3e755a045d80bcf946ef0e0d1c981e193ef46b1a2c53c1b665cc2b474ea6cc8b  interfaces/core/IPriceProvider.sol
480104c4262c9dc30d680b398bf8f60b804dd1ebb11f36d26b318fada5fe335f  interfaces/utils/IUniswapV3CrossPoolOracle.sol
07032d2973ac7011bf8d547b4ff893cc75a03642cd2cefd54913d19bc6bf8d81  core/ChainlinkMainnetPriceProvider.sol
d5f1ba0339f3a1d2a2e4f338a2fea8313d03a660f7848cdba0ca5c0ab3e8f6de  core/ChainlinkAvalanchePriceProvider.sol
c80afcb054150da02f5277ab9b30fe9301a1e578aab55e87fd3e85da54372ee8  core/ChainlinkArbitrumPriceProvider.sol
b0da00ca9bde38b67dbc69b23fbde3dd218b4fee96cc5da50cd75cda6e729a70  core/ChainlinkPolygonPriceProvider.sol
85bfc35c2b9486b33e37eed1d86860126fe3624bfb4006e6b3e74d86a8151659  core/UniswapV2LikePriceProvider.sol
ef75c88d77a8113b36d65b6694fb30ce51fa22635559e7294033d68cc0cb53be  core/UniswapV3PriceProvider.sol
8d0a548f9f31d4b3be1d7849424acdbc295622ece073390f360104141e565aec  core/UmbrellaPassportPriceProvider.sol
f23ca66349adde864695168b9cee0fdcadf0add9acd521f59aca8651574febcb  core/UmbrellaPriceProvider.sol
22e1c46c0ac278b85b98671c2fc54fa1fbcf2f197842753f29dc36d9daee7c91  core/PriceProvidersAggregator.sol
82d5b199cb22f9d02ea88102581f33ea034cd48c37482dfe224a6d43ce07d673  core/ChainlinkPriceProvider.sol
5467738274737c7cd6b5fcffcfa7bd8b82458b420304061e001ddf8a91e345dd  utils/UniswapV3CrossPoolOracle.sol
20500e9d01139d7467f7132bf82a3ce38e68a5887210d66a8dd88aa1eedca919  access/Governable.sol
```

3. Summary of Findings

Id	Title	Total Risk	Fixed
ONE-1	Chainlink feed assumed to always have 8 decimals	Medium	✘
ONE-2	Attackers can add arbitrary price oracles	Medium	✘
ONE-3	Stablecoin used as price reference could deviate from $1 USD peg	Medium	✘
ONE-4	Duplicated check in the _addOracleFor function	Low	✘
ONE-5	Attackers can obtain assets sent by error to Exchange contract	Medium	✘
ONE-6	swapExactInput and swapExactOutput callers unprotected from configurable max slippage changes	Low	✘
ONE-7	Missing pairFor error return value validation	Info	✘
ONE-8	Incorrect function parameter verification	Info	✘
ONE-9	quote's _lastUpdatedAt return value timestamp has different meaning depending on the type of oracle	Info	✘

4. Detailed Findings

ONE-1	Chainlink feed assumed to always have 8 decimals	
Total Risk Medium	Impact High	Location `ChainlinkPriceProvider.sol`
Fixed ✘	Likelihood Medium	

Description

The code assumes all Chainlink feeds use 8 decimals precision, and a hardcoded constant (10) is used to adjust the rates to 18 decimals precision. This could result in invalid calculations that harm the protocol if any feed has a different number of decimals.

```
contract ChainlinkPriceProvider is IChainlinkPriceProvider, Governable {
   /**
    * @notice Used to convert 8-decimals from Chainlink to 18-decimals values
    */
   uint256 public constant TEN_DECIMALS = 1e10;
```

For instance, Coinspect observed that the “Calculated SAVAX / USD” price feed for the in the Avalanche network has 18 decimals instead of 8: https://docs.chain.link/docs/avalanche-price-feeds/

Recommendation

Use Chainlink `decimals()` function to retrieve the number of decimals used.

Status

Open.

ONE-2 Attackers can add arbitrary price oracles

Total Risk	Impact	Location
Medium	High	`UniswapV2LikePriceProvider.sol`
Fixed ✘	Likelihood Low	

Description

Attackers can call the `updateOrAdd` function with an attacker-controlled non-verified `UniswapV2Pair` address to add an oracle they control to the contract's `oracles` mapping.

```
function updateOrAdd(IUniswapV2Pair pair_) external override {
    updateOrAdd(pair_, default Swap Period);
}

/// @inheritdoc IUniswapV2LikePriceProvider
function updateOrAdd(IUniswapV2Pair pair_, uint256 twapPeriod_) public override {
```

These functions rely on the private `_addOracleFor` function:

```
function _addOracleFor(IUniswapV2Pair pair_, uint256 twapPeriod_) private {
    if (hasOracle(pair_, twapPeriod_)) return;

    (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) = pair_.getReserves();

    require(_reserve0 != 0 && _reserve1 != 0, "no-reserves");

    oracles[pair_][twapPeriod_] = Oracle({
        token0: pair_.token0(),
        token1: pair_.token1(),
        price0CumulativeLast: pair_.price0CumulativeLast(),
        price1CumulativeLast: pair_.price1CumulativeLast(),
        blockTimestampLast: _blockTimestampLast,
        price0Average: uint112(0).encode(),
        price1Average: uint112(0).encode()
    });
```

Prices obtained from these oracles will be updated using the information they provide.

However, all the public functions that return a price quote (e.g. `quote()`) access the oracles mapping with an oracle address obtained from the `pairFor` function using the Uniswap factory contract configured in the `UniswapV2LikePriceProvider` contract constructor.

As a consequence, the attacker-controlled oracles could only be accessed by reading the contract storage and ignoring the intended public interface. For example, off-chain code could trust Vesper's price provider contract and/or rely on prices retrieved from the **public** `oracle` storage variable instead of consulting the intended public functions.

Because this issue is exploitable under very specific circumstances, the likelihood of this issue is considered low.

Recommendation

Only add oracles using pair contract addresses obtained by calling `pairFor` and the trusted Uniswap factory.

Status

Open.

ONE-3 Stablecoin used as price reference could deviate from $1 USD peg

Total Risk	Impact	Location
Medium	High	`periphery/Oracle.sol`
Fixed ✘	Likelihood Low	

Description

When not using Chainlink oracles, a USD pegged stable coin is used as a proxy of the USD to convert between assets. This is risky as stablecoins peg can be lost in extreme volatility scenarios and be abused to perform trades that could harm the platform.

This fact is acknowledged by a comment in the source code:

```
contract Oracle is IOracle, Governable {
   uint256 public constant USD_DECIMALS = 18;

   /**
    * @notice A stable coin to use as USD price reference if provider is UniV2 or UniV3 (optional)
    * @dev Stable coin may lose pegging on-chain and may not be equal to $1.
    */
   address public usdEquivalentToken;
```

Recommendation

Actively monitor the stablecoin used by the oracle to be able to promptly pause operations and/or the oracle relying on it.

Status

Open.

ONE-4 Duplicated check in the _addOracleFor function

Total Risk	Impact	Location
Low	Low	`UniswapV2LikePriceProvider.sol`
Fixed ✗	Likelihood Low	

Description

The private `_addOracleFor` function rechecks if there is a registered oracle for the pair and twap period even though this check has been already done in the `updateOrAdd` function, its only caller.

```
function updateOrAdd(IUniswapV2Pair pair_, uint256 twapPeriod_) public override {
    if (oracles[pair_][twapPeriod_].blockTimestampLast == 0) {
        _addOracleFor(pair_, twapPeriod_);
    }
    _updateIfNeeded(pair_, twapPeriod_);
}

function _addOracleFor(IUniswapV2Pair pair_, uint256 twapPeriod_) private {
    if (hasOracle(pair_, twapPeriod_)) return;
```

```
function hasOracle(IUniswapV2Pair pair_, uint256 twapPeriod_) public view override returns (bool)
{
    return oracles[pair_][twapPeriod_].blockTimestampLast > 0;
}
```

Recommendation

Remove the duplicated check to save gas. Also, consider using the `hasOracle` function in the `updateOrAdd` function for consistency's sake.

Status

Open.

ONE-5 Attackers can obtain assets sent by error to Exchange contract

Total Risk	Impact	Location
Medium	High	`UniswapV2LikeExchange.sol`
Fixed	Likelihood	
✘	Low	

Description

The swapExactOutput can be abused to retrieve any funds available in the contract.

This is intended to allow swappers to obtain back their own tokens if less than the maximum amount was utilized for the exchange. However, this feature is implemented by sending back all available balance in the contract:

```
  function swapExactOutput(
      address[] calldata path_,
      uint256 amountOut_,
      uint256 amountInMax_,
      address inSender_,
      address outRecipient_
  ) external returns (uint256 _amountIn) {
      IERC20(path_[0]).safeApprove(address(router), amountInMax_);
      _amountIn = router.swapTokensForExactTokens(amountOut_, amountInMax_, path_, outRecipient_,
block.timestamp)[0];
      // If swap end up costly less than _amountInMax then return remaining
      uint256 _remainingAmountIn = IERC20(path_[0]).balanceOf(address(this));
      if (_remainingAmountIn > 0) {
          IERC20(path_[0]).safeTransfer(inSender_, _remainingAmountIn);
      }
  }
```

Even though this contract should not hold funds, if a user transfers tokens to the contract by mistake, any other user would be able to extract them.

Recommendation

Only transfer back the amount corresponding to the caller.

Status

Open.

ONE-6	swapExactInput and swapExactOutput callers unprotected from configurable max slippage changes	
Total Risk **Low**	Impact Low	Location `Swapper.sol`
Fixed ✘	Likelihood Low	

Description

The Swapper contract users must trust the contract configurable slippage not to change when calling the `swapExactInput` and `swapExactOutput` functions.

The functions obtain the best quote available from the configured exchanges, and make the swap with a configured maximum slippage. The maximum slippage can be modified anytime by the Governor role.

Because the functions do not accept a minimum/maximum expected amount of tokens, the call could result in an unfavorable exchange for the users if the maximum slippage is changed before a user transaction is mined.

Recommendation

Consider adding a maximum tokens input or a minimum expected output to the functions.

Status

Open.

ONE-7 Missing pairFor error return value validation

Total Risk	Impact	Location
Info	-	`UniswapV2LikePriceProvider.sol`
Fixed	**Likelihood**	
✘	-	

Description

The return value from the UniswapV2Factory `getPair` function is not checked to be not zero and execution continues using the zero address as a valid pair address. While sometimes the code is able to catch the error later on, this could also result in unexpected results being returned to the user, or an exploitable situation could be introduced in the future.

```
    function pairFor(address token0_, address token1_) public view override returns (IUniswapV2Pair
_pair) {
        _pair = IUniswapV2Pair(IUniswapV2Factory(factory).getPair(token0_, token1_));
    }
```

Quoting Uniswap's documentation for the `pairFor` function: *Returns the address of the pair for tokenA and tokenB, if it has been created, else address(0) (`0x0000000000000000000000000000000000000000`)*.

This issue is not currently exploitable.

Recommendation

Revert the transaction as soon as possible when a non-existing token pair operation is attempted.

Status

Open.

ONE-8 Incorrect function parameter verification

Total Risk	Impact	Location
Info	-	UniswapV3CrossPoolOracle.sol
Fixed ✘	Likelihood -	

Description

The require check in the `assetToAssetThruRoute` function is not necessary. `poolFees_` length is checked at runtime since it is declared as having length two.

```
function assetToAssetThruRoute(
    address tokenIn_,
    uint256 amountIn_,
    address tokenOut_,
    uint32 twapPeriod_,
    address routeThruToken_,
    uint24[2] memory poolFees_
) public view returns (uint256 amountOut) {
    require(poolFees_.length <= 2, "uniV3CPOracle: bad fees length");
    uint24 _pool0Fee = poolFees_[0];
    uint24 _pool1Fee = poolFees_[1];
```

Recommendation

Remove the unnecessary require call.

Status

Open.

ONE-9	quote's _lastUpdatedAt return value timestamp has different meaning depending on the type of oracle	

Total Risk
Info
	Impact
	-
		Location
		`UniswapV3PriceProvider.sol`
		`UniswapV2LikePriceProvider.sol`
		`SwapperOracle.sol`

Fixed ✘	Likelihood -	

Description

The return value of the `quote*` functions family has a different meaning depending on the kind of underlying oracle being consulted. As a consequence, prices provided by Uniswap based oracles will not be considered stale even if the last observation used to update the price is not fresh.

For Uniswap based oracles (V2 and V3) the timestamp represents the last time the price information was updated by an AMM interaction. This might not correspond to the last time the price was updated in the AMM.

In particular, the UniswapV3 based price provider provides callers with a last update timestamp that is the current block timestamp, making it impossible to detect and react to an stale oracle. To provide the same interface for the quote function, the current block timestamp is returned to the caller as the last update timestamp. This is misleading because the oracle consumer will always believe the oracle was updated during the current block.

```
function quote(
    address tokenIn_,
    address tokenOut_,
    uint24 poolFee_,
    uint32 twapPeriod_,
    uint256 amountIn_
) public view override returns (uint256 _amountOut, uint256 _lastUpdatedAt) {
    if (tokenIn_ == tokenOut_) {
        return (amountIn_, block.timestamp);
    }

    [...]
```

```
        _lastUpdatedAt = block.timestamp;
    }
```

The UniswapV2 based price provider returns the timestamp of the last time the price was calculated. In this case, the timestamp is obtained from the `UniswapV2OracleLibrary currentCumulativePrices` function call. This does not represent the time of the last observation performed by the oracle either.

Recommendation

Clearly document that the returned value has a different meaning depending on which kind of oracle is being consulted. Further evaluate how this could impact the stale detection mechanism implemented in the SwapperOracle.sol contract.

Status

Open

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.