Exactly

Smart Contract Audit

Exactly Protocol

Smart Contract Audit

V221028

Prepared for Exactly • October 2022

1. Executive Summary

2. Assessment and Scope

3. Summary of Findings

4. Detailed Findings

EXA-36 Chainlink response liveliness not checked

EXA-37 PriceFeedWrapper rate return value not checked

EXA-38 Invalid price feeds could be set while enabling new markets

EXA-39 mockStETH implementation allows public minting

5. Disclaimer

1. Executive Summary

In **October 2022**, **Exactly** engaged Coinspect to perform an incremental source code review of a limited set of modifications performed to **Exactly Protocol** since the previous audit. The objective of the project was to evaluate the security of the smart contracts, particularly the commits shown in the Assessment section below.

The changes introduced by these commits include:

1. How prices are obtained from external oracles such as Chainlink.
2. The new ability to apply a rate to the asset price by means of calling another external contract, intended to be used with the Lido staking protocol (Lido implementation was not in scope).
3. USD is no longer used as the base price.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
Open **0**	Open **0**	Open **0**
Fixed 0	Fixed 0	Fixed 1
Reported 0	Reported 2	Reported 1

Coinspect identified 2 medium-risk issues related to an unchecked staleness for the oracle's price feed data and the performance of unchecked external `staticcalls`. Chainlink oracle's data might be stale yet used anyways across the different markets affecting borrows, deposits, repayments and liquidations. The Exactly team is aware of the potential risk and willing to accept it in order to optimize gas consumption as further explained in the finding. Also, the implementation of the rate wrapper retriever contract performs an external `staticcall` whose success and return data are not checked.

2. Assessment and Scope

The audit started on **Oct 24, 2022** and was conducted on the **dev** branch of the git repository at https://github.com/exactly-protocol/protocol/ as of the commits shown below.

The audit focused on the following commits:

- `9eaee805a8c42322a646749b105ef884c78dc531`
- `49d607d5db7255dcbf0a238c6090436b01269a9a`
- `1d9a1b42096ca3020ed29ce11926828fe6ff5e79`
- `4430015f416484a7110fab11bcac63de8b8039c6`
- `744669bdb9c53d8109065514a8667aa838b83736`

The new pull requests were easy to read and the code was understandable. However, some public variables as well as contracts currently do not have Natspec comments which play a key role in the UX (for example, they are automatically retrieved from the contract by Etherscan and shown on its getters and setters). Coinspect suggests adding documentation for mock contracts as well to prevent their reusability in the production phase (either from Exactly and third-parties) as they might have critical access control issues (see **EXA-39**).

As for breaking changes introduced by the above mentioned commits in comparison with the last audit, there are critical aspects that were modified:

1. The base price is no longer USD and changed to ETH in order to reduce the queries to the oracle reducing the consumption of gas.
2. An oracle contract is no longer used and the `Auditor` processes the external oracle calls. This required changing the structure of the markets by adding the `priceFeed` parameter to each one of them.
3. A price feed wrapper contract was introduced that allows calling an external source to obtain a rate then applied to the price.

On the contrary of the older oracle implementation, the new oracle retrieval implementation located in the `Auditor` contract does not take into account an eventual failure of Chainlink's data feed that might lag the retrieved value. Although several DeFi protocols opt not to perform that check, Coinspect suggests being on the "safe side" by checking for data staleness (see **EXA-36**). Also, as the retrieval for each market is done in the `Auditor`, newly enabled markets require a `priceFeed` value which validity is not checked while enabling new markets and it is only checked once it is modified. This scenario can cause severe misalignments on every market as arbitrage scenarios as unfair positions could be opened and closed within that time-range (see **EXA-38**).

A `PriceFeedWrapper` contract was recently added in order to retrieve the rate of stETH from a foreign contract. Although its implementation was provided, it lacks a threat model as well as a concrete implementation of the pointed wrapper. Coinspect recommends auditing all wrappers before deployment because they perform external interactions whose threats remain unknown. Regarding the `PriceFeedWrapper` implementation, a `staticcall` is performed to the external wrapper which does not check both that the boolean return is successful and that the retrieved data is between reasonable boundaries (see **EXA-37**).

The Exactly team stated to Coinspect in a communication that this feature is intended to be used with Lido staking protocol, to obtain the stETH / ETH rate by calling `Lido.getPooledEthByShares()` selector. As the return values of the mentioned function depend on protocol states (mainly on the supply of shares), **Coinspect suggests performing a deeper revision to check if the return value could be in some way manipulated which would impact directly on Exactly.** As reviewing Lido's code is out of scope for this audit, we recommend checking under which scenarios the returned value of `getPooledEthByShares()` (called by `PriceFeedWrapper.latestAnswer()`) could be manipulated and/or front-runned.

3. Summary of Findings

Id	Title	Total Risk	Fixed
EXA-36	Chainlink response liveliness not checked	Medium	!
EXA-37	PriceFeedWrapper rate return value not checked	Medium	!
EXA-38	Invalid price feeds could be set while enabling new markets	Low	✔
EXA-39	mockStETH implementation allows public minting	Info	✔

4. Detailed Findings

EXA-36 Chainlink response liveliness not checked

Total Risk	Impact	Location
Medium	High	`Auditor.sol`
Fixed !	Likelihood Low	

Description

No liveness checks are performed while retrieving oracle data. As a result, prices could be outdated yet used anyways affecting deposits, borrows, repayments, and any other source that relies on Chainlink's prices.

Although liveness checks were performed on older codebase versions, these were removed as a gas optimization. The price retrieval is needed by critical paths in the implementation such as the liquidity calculation loop.

Lags in Chainlink price updates, though not common, have been exploited in the past, for example in 2020 where $4.5 million worth of DAI were left unbacked by any collateral on Maker DAO when full blocks prevented Chainlink price updates from being mined.

Recommendation

Even though this oracle price retrieval pattern is common among other protocols (e.g. Notional, Euler, AAVE), Coinspect suggests re-adding the liveness checks in order to protect users in the worst case scenarios.

Status

Exactly stated they know and accept the risk:

We realized Euler's protocol also checks for the price this way. Considering a low minimum timelock delay (1 day) and an upgradable Auditor, we can act fast in case prices are not being updated as they should. We also believe that Chainlink offers robust and historically stable price feeds, even more on Mainnet for high liquid assets such as WBTC, ETH, DAI... and if there's a problem with any oracle we'll find out asap (we are working on monitoring integrations and scripts). On the other hand, another difference with checking liveness is that transactions would revert in case of outdated updateTimes but as a downfall that can also prevent liquidations from happening. These are mainly the reasons why we agree on assuming the risk and lowering gas costs on the trade off.

Coinspect auditors reviewed this answer and believe:

1. Regarding the one day timelock, the mentioned link in this issue depicts that a lag of 6 hours was just enough to break the functioning of Maker DAO's liquidations.
2. It is true that reverting while being outdated and suddenly operating with a new price can lead to a considerable step change. However, an updated price should be used when handling debt and deposits as well as liquidations. Preventing liquidations that may arise due to stale oracle data is exactly what should happen. No liquidations, borrows and deposits should occur when the price is outdated as it is an unfair scenario for users and the protocol that leads to arbitrage opportunities whose impact and side effects are unknown. If the sudden step price change is meant to be prevented while Chainlink is not responding, a backup oracle should be implemented instead.

Finally, Exactly team responded:
We believe that the example that was quoted happened in a really early state of DeFi. Historically, Chainlink has already proven stability during other black swan events and volatility in gas prices.
We also consider that this implementation is only going to be used for Ethereum, whereas for L2's deployments we will reassess the risk.
For these reasons and the ones already mentioned in the status section, we will not take additional actions.

EXA-37 PriceFeedWrapper rate return value not checked

Total Risk	Impact	Location
Medium	High	`PriceFeedWrapper.sol`
Fixed !	Likelihood Low	

Description

The data retrieval from the rate conversion wrapper does not check the retrieved price and the success condition. As a result, the `PriceFeedWrapper.latestAnswer()` could return negative or invalid data yet used anyways across the market.

The mentioned function has the following implementation:

```
 function latestAnswer() external view returns (int256) {
    int256 mainPrice = mainPriceFeed.latestAnswer();

    (, bytes memory data) =
address(wrapper).staticcall(abi.encodeWithSelector(conversionSelector, baseUnit));
    uint256 rate = abi.decode(data, (uint256));

    return int256(rate.mulDivDown(uint256(mainPrice), baseUnit));
  }
```

On the other hand, `Auditor.assetPrice()` is implemented as follows:

```
  function assetPrice(IPriceFeed priceFeed) public view returns (uint256) {
    if (address(priceFeed) == BASE_FEED) return basePrice;

    int256 price = priceFeed.latestAnswer();
    if (price <= 0) revert InvalidPrice();
    return uint256(price) * baseFactor;
  }
```

The low level `staticcall` function has two returns, a `boolean success` and `bytes data`. Currently, the decoded rate has no rules as the price has in `assetPrice()`. Also, there are no checks that ensure that the boolean return is true.

Recommendation

Check both the boolean return and the retrieved rate if possible.

This is particularly important if in the future rates are retrieved from other sources instead of the Lido contract only.

Status

Exactly stated they know and accept the risk:

Given that the low level staticcall fails (bool success = false), the decoded data will be 0, hence the latestAnswer() return value will also be 0. Then the Auditor will not consider this as a valid price due to this check.
We've made research about Chainlink's bad/invalid price answers and the standard value for these cases is also 0.
On top of this, for a negative Chainlink price to be casted as positive and not to revert, this price has to be equal to -1 and the rate value should be equal to only 1 unit. All other cases for negative prices will revert with an overflow when multiplied by the rate, as seen in this test.
Due to these reasons we are not going to take additional actions.

Coinspect auditors reviewed this answer and would like to add that it is a good practice catching errors on site instead of relying on another layer to catch it. Unchecked `staticcalls` that revert with a return error string will return non-zero data that could negatively impact the markets that query from the price feed wrapper. This could happen if the `PriceFeedWrapper` contract is used with different data feeds that return revert strings on failure. It is worth noting the generic wrapper design suggests it could be used with other feeds in the future. Then, it is strongly suggested to make external low level calls bulletproof by checking their boolean return.

An example of this potential issue is shown below:

Output on revert
3963877391197344453575983046348115674221700746820753546331534351508065746944

```
pragma solidity 0.8.17;

///@notice Mock contracts. Do not use them in production.
contract MockPriceFeedWrapper {
  address public wrapper;

  constructor(address _wrapper){
      wrapper = _wrapper;
  }

  function latestAnswer() external view returns (int256) {
    (, bytes memory data) =
       address(wrapper).staticcall(abi.encodeWithSignature("fakePrice()"));
    uint256 rate = abi.decode(data, (uint256));
    return int256(rate);

  }
}

contract MockFeed {
    function fakePrice() external view returns(uint256){
        require(block.timestamp % 2 == 0, "A revert string");
        return 1;
    }
}
```

EXA-38 Invalid price feeds could be set while enabling new markets

Total Risk	Impact	Location
Low	Medium	`Auditor.sol`
Fixed ✓	Likelihood Low	

Description

The checks performed while calling `Auditor.setPriceFeed()` are not performed while enabling new markets. As a consequence, the feeds for recently enabled markets could return outscaled or invalid prices affecting the debt and deposits across the protocol.

Invalid price feed checks are included while calling `setPriceFeed()`:

```
  function setPriceFeed(
               Market market,
               IPriceFeed priceFeed
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (address(priceFeed) != BASE_FEED && priceFeed.decimals() != priceDecimals) revert
InvalidPriceFeed();
    markets[market].priceFeed = priceFeed;
    emit PriceFeedSet(market, priceFeed);
  }
```

However, while enabling new markets, the check that reverts with `InvalidPriceFeed()` made on `setPriceFeed()` is not performed:

```
 function enableMarket(
    Market market,
    IPriceFeed priceFeed,
    uint128 adjustFactor,
    uint8 decimals
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (market.auditor() != this) revert AuditorMismatch();

    if (markets[market].isListed) revert MarketAlreadyListed();

    markets[market] = MarketData({
      isListed: true,
      adjustFactor: adjustFactor,
      decimals: decimals,
      index: uint8(marketList.length),
      priceFeed: priceFeed
    });

    marketList.push(market);

    emit MarketListed(market, decimals);
    emit PriceFeedSet(market, priceFeed);
    emit AdjustFactorSet(market, adjustFactor);
  }
```

Recommendation

Include invalid price feed checks while enabling new markets.

Status

Acknowledged.

This issue is similar in nature to the previously reported **EXA-22** issue.

This issue was addressed in commit:
https://github.com/exactly-protocol/protocol/commit/2152f33fd2c7a583a27c612fe6f72e6d30a631e1.

EXA-39 mockStETH implementation allows public minting

Total Risk	Impact	Location
Info	-	`MockStETH.sol`
Fixed ✔	Likelihood -	

Description

The `mockStETH` contract implements a non-access controlled mint function. In the event of mistakenly using that contract on a production phase (either by Exactly or by another third party), anyone could mint unlimited tokens.

```
function mint(address to, uint256 value) public virtual {
  _mint(to, value);
}
```

It is worth noting this contract is only intended for testing purposes.

Recommendation

Document that the contract is meant to be used only for the development phase.

Status

This issue was addressed by commit:
https://github.com/exactly-protocol/protocol/commit/6adf07c3238b026173803ed7ea866deb609515eb.

This issue is not currently exploitable.

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.