HALBORN

# Fetch Oracle - EVM Contracts

## Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 7th, 2023 - November 6th, 2023**

Visit: **Halborn.com**

# DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE |
|---|---|---|
| 0.1 | Document Creation | 08/23/2023 |
| 0.2 | Document Updates | 08/24/2023 |
| 0.3 | Draft Version | 08/31/2023 |
| 0.4 | Draft Review | 09/01/2023 |
| 0.5 | Draft Review | 09/01/2023 |
| 1.0 | Document Updates | 11/01/2023 |
| 1.1 | Document Updates | 11/02/2023 |
| 1.2 | Document Updates Review | 11/13/2023 |
| 1.3 | Document Updates Review | 11/13/2023 |
| 2.0 | Remediation Plan | 12/04/2023 |
| 2.1 | Remediation Plan Review | 12/05/2023 |
| 2.2 | Remediation Plan Review | 12/06/2023 |

# CONTACTS

| CONTACT | COMPANY | EMAIL |
| --- | --- | --- |
| Rob Behnke | Halborn | Rob.Behnke@halborn.com |
| Steven Walbroehl | Halborn | Steven.Walbroehl@halborn.com |
| Gabi Urrutia | Halborn | Gabi.Urrutia@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

Fetch Oracle engaged Halborn to conduct a security assessment on their smart contracts. The security assessment was scoped to the smart contracts provided. Commit hashes and further details can be found in the Scope section of this report.

The fetch oracle project is a decentralized oracle meant to be run on pulse chain, forked from the tellor oracle protocol.

## 1.2 ASSESSMENT SUMMARY

Halborn was provided 7 weeks across two different periods for the engagement:
- August 7th - August 29th
- October 2nd - November 6th

A full-time security engineer was assigned to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were addressed and accepted by Fetch Oracle. The main ones were the following:

- The client accepted the risk of a denial of service over disputes.
- The client accepted the risk of funding feeds or creating single tips, leading to a denial of service.

## 1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Testnet deployment (Foundry, Brownie).

# 2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

## 2.1 EXPLOITABILITY

### Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

### Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

### Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

### Metrics:

| Exploitability Metric ($m_E$) | Metric Value | Numerical Value |
|---|---|---|
| Attack Origin (AO) | Arbitrary (AO:A)<br>Specific (AO:S) | 1<br>0.2 |
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>0.33 |

Exploitability $E$ is calculated using the following formula:

$$E = \prod m_e$$

## 2.2 IMPACT

### Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

### Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

### Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

### Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

### Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

| Impact Metric ($m_I$) | Metric Value | Numerical Value |
|---|---|---|
| Confidentiality (C) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Integrity (I) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Availability (A) | None (A:N) | 0 |
| | Low (A:L) | 0.25 |
| | Medium (A:M) | 0.5 |
| | High (A:H) | 0.75 |
| | Critical | 1 |
| Deposit (D) | None (D:N) | 0 |
| | Low (D:L) | 0.25 |
| | Medium (D:M) | 0.5 |
| | High (D:H) | 0.75 |
| | Critical (D:C) | 1 |
| Yield (Y) | None (Y:N) | 0 |
| | Low (Y:L) | 0.25 |
| | Medium: (Y:M) | 0.5 |
| | High: (Y:H) | 0.75 |
| | Critical (Y:H) | 1 |

Impact $I$ is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

## 2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

| Coefficient ($C$) | Coefficient Value | Numerical Value |
|---|---|---|
| Reversibility ($r$) | None (R:N)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>1 |

Severity Coefficient $C$ is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score $S$ is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

| Severity | Score Value Range |
|---|---|
| Critical | 9 - 10 |
| High | 7 - 8.9 |
| Medium | 4.5 - 6.9 |
| Low | 2 - 4.4 |
| Informational | 0 - 1.9 |

## 2.4 SCOPE

**1. IN-SCOPE REPOSITORIES & COMMIT IDS :**

Code repositories:

1. Fetch Oracle Governance

- Repository: fetchoracle/governance
- Commit ID: d82443ca59d47e00d066a14f82f1ca858da89077
- Smart contracts in scope:
    1. Governance.sol (contracts/Governance.sol)

2. Fetch Oracle Autopay

- Repository: fetchoracle/autoPay
- Commit ID: 6a5b510898c3a6f698054356c4448946f59fcdc3
- Smart contracts in scope:
    1. Autopay.sol (contracts/Autopay.sol)
    2. QueryDataStorage.sol (contracts/QueryDataStorage.sol)

3. Fetch Oracle Fetch Flex

- Repository: fetchoracle/fetchFlex
- Commit ID: 25adf745995f50b31d30078f90727cf1ef65e601
- Smart contracts in scope:
    1. FetchFlex.sol (contracts/FetchFlex.sol)

4. Fetch Oracle Fetch Token

- Repository: fetchoracle/FETCH-Token
- Commit ID: 0e6ea5abdbc3ae06bb6e27074343b2c391b7dc48

- Smart contracts in scope:
    1. FetchToken.sol (contracts/FetchToken.sol)

5. Fetch Oracle Using Fetch

- Repository: fetchoracle/usingfetch
- Commit ID: 381dcdbeeedacc653c9f5f2f800f65f277aa4e9b
- Smart contracts in scope:
    1. UsingFetchUpgradeReady.sol (contracts/UsingFetchUpgradeReady.sol)

6. Secondary Oracle

- Repository: fetchoracle/secondaryOracle
- Commit ID: 3d1cc0b
- Smart contracts in scope:
    1. SecondaryOracle.sol (contracts/SecondaryOracle.sol)
    2. ChainlinkHelper.sol (contracts/ChainlinkHelper.sol)
    3. TwapHelper.sol (contracts/TwapHelper.sol)

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

**2. REMEDIATION COMMIT IDs :**

- 608339b7fe47043852b29474fbd2ef2412ba470b
- f7ae06d1728b5665c5e35f5edf4ec6c52eeb2a8a
- 5ac3f75e51cbb392825c5dd1d039bd7d02a5f97b
- 49f6f4022250b972fcf419c5c0f55e5457b74401
- a853d3770cafd19a55fbee74e6b83b20e7db2471

# 3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 0 | 5 | 2 | 1 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| (HAL-01) EXECUTING VOTES IN GOVERNANCE CAN LEAD TO DENIAL OF SERVICE OVER A DISPUTE | Medium (4.8) | RISK ACCEPTED |
| (HAL-02) FUNDING FEEDS OR CREATING SINGLE TIPS CAN LEAD TO DENIAL OF SERVICE | Medium (5.6) | RISK ACCEPTED |
| (HAL-03) FEEDS WILL APPEAR AS FUNDED EVEN IF THEY DONT HAVE ENOUGH BALANCE TO COVER THE REWARDS | Medium (5.0) | RISK ACCEPTED |
| (HAL-04) MISSING STORAGE GAPS | Medium (5.2) | SOLVED - 12/03/2023 |
| (HAL-05) CHAINLINK HELPER CAN RETURN INCORRECT PRICES | Medium (6.7) | SOLVED - 12/03/2023 |
| (HAL-06) IMPLEMENTATION CONTRACT CAN BE INTIIALIZED | Low (2.5) | SOLVED - 12/03/2023 |
| (HAL-07) REDUNDANT DATA OVERWRITING | Low (2.5) | RISK ACCEPTED |
| (HAL-08) FLOATING PRAGMA | Informational (0.0) | ACKNOWLEDGED |

# FINDINGS & TECH DETAILS

# 4.1 (HAL-01) EXECUTING VOTES IN GOVERNANCE CAN LEAD TO DENIAL OF SERVICE OVER A DISPUTE - MEDIUM (4.8)

Description:

The executeVote() function in the Governance contract executes the votes and transfers the corresponding balances to the initiator and reporter.

In order to perform these operations, it iterates over the voteRounds array, retrieving information from each vote round. However, there is no hardcoded limit to the length of this array and the operation is not performed in batches.

This means, that with enough vote rounds in the array, it would be possible for the executeVote function to run out of gas.

Code Location:

**Listing 1: src/governance/Governance.sol (Line 253)**

```
253 for (
254     _i = voteRounds[_thisVote.identifierHash].length;
255     _i > 0;
256     _i--
257 ) {
258     _voteID = voteRounds[_thisVote.identifierHash][_i - 1];
259     _thisVote = voteInfo[_voteID];
260     // If the first vote round, also make sure to transfer the
 ↳ reporter's slashed stake to the initiator
261     if (_i == 1) {
262         token.transfer(
263             _thisVote.initiator,
264             _thisDispute.slashedAmount
265         );
266     }
```

```
267     token.transfer(_thisVote.initiator, _thisVote.fee);
268 }
```

Proof Of Concept:

**Listing 2: test/Governance.t.sol**

```
1  function test_Governance_ExecuteVotes_DOS() public {
2      bytes memory queryData = abi.encode(
3          "SpotPrice",
4          abi.encode("btc", "usd")
5      );
6
7      token.increaseAllowance(address(autoPay), 100 ether);
8
9      autoPay.setupDataFeed(
10         keccak256(queryData),
11         1 ether, // Reward / 1 TRB
12         block.timestamp + 1 days, // Start Time / Tomorrow
13         1 hours, // Interval / We want to retrieve price once per
↳ hour.
14         5 minutes, // Window / We want it reported within 5
↳ minutes.
15         100, // Price Threshold
16         10, // Reward Increase per second.
17         queryData,
18         100 ether
19     );
20
21     token.increaseAllowance(address(flex), 1 ether);
22     flex.depositStake(1 ether);
23
24     skip(1 days);
25
26     uint256 timestamp = block.timestamp;
27
28     flex.submitValue(
29         keccak256(queryData),
30         abi.encodePacked(uint256(1 ether)),
31         0,
32         queryData
33     );
```

```
34
35      token.increaseAllowance(address(governance), 1000000 ether);
36
37      for (uint i; i < 300; ++i) {
38          governance.beginDispute(keccak256(queryData), timestamp);
39          skip(6 days);
40          governance.tallyVotes(i + 1);
41      }
42
43      skip(1 days);
44      governance.executeVote(300);
45  }
```

BVSS:

**AO:A/AC:M/AX:H/C:N/I:C/A:C/D:C/Y:C/R:N/S:C (4.8)**

Recommendation:

Implement a batch system so if the array surpasses a certain length, votes are executed in batches.

Remediation Plan:

**RISK ACCEPTED**: The Fetch Oracle team accepted the risk as they considered that based on the extremely high-cost and extended time needed to dispute and vote on issues, it is highly unlikely for the contract to end up in this state. It wouldn't be profitable for anyone to attempt this, so this scenario is extremely unlikely.

## 4.2 (HAL-02) FUNDING FEEDS OR CREATING SINGLE TIPS CAN LEAD TO DENIAL OF SERVICE - MEDIUM (5.6)

Description:

Some external view functions within the AutoPay contract such as getFundedFeedDetails() or getFundedSingleTipsInfo loop through whole arrays saved as state variables.

If this arrays gets too big, for example, by getting enough funded feeds it is possible to reach a point where the function may run out of gas before ending execution.

However, because these are external view functions this would only happen if called by an external contract, as view functions when called through an RPC are processed by the RPC and do not consume any gas.

Code Location:

**Listing 3: src/autopay/Autopay.sol (Line 422)**

```
414 function getFundedFeedDetails()
415     external
416     view
417     returns (FeedDetailsWithQueryData[] memory)
418 {
419     bytes32[] memory _feeds = this.getFundedFeeds();
420     FeedDetailsWithQueryData[]
421         memory _details = new FeedDetailsWithQueryData[](_feeds.
 ↳ length);
422     for (uint256 i = 0; i < _feeds.length; i++) {
423         FeedDetails memory _feedDetail = this.getDataFeed(_feeds[i
 ↳ ]);
424         bytes32 _queryId = this.getQueryIdFromFeedId(_feeds[i]);
425         bytes memory _queryData = queryDataStorage.getQueryData(
 ↳ _queryId);
426         _details[i].details = _feedDetail;
```

```
427             _details[i].queryData = _queryData;
428         }
429         return _details;
430     }
```

BVSS:

**AO:A/AC:M/AX:M/C:N/I:C/A:C/D:N/Y:N/R:N/S:U (5.6)**

Recommendation:

Allow to specify a range in the index to retrieve when calling the function.

Remediation Plan:

**RISK ACCEPTED**: The issue would only affect the external protocols calling this oracle from another smart contract. Therefore, the Fetch Oracle team accepted the risk as they considered that the problem does not directly affect the Fetch Protocol. As these are external view functions meant to be called directly through an RPC node without using any gas.

## 4.3 (HAL-03) FEEDS WILL APPEAR AS FUNDED EVEN IF THEY DONT HAVE ENOUGH BALANCE TO COVER THE REWARDS - MEDIUM (5.0)

Description:

When funding a feed in the Autopay contract through the fundFeed and tip functions, the feed / query will be pushed to the feedsWithFunding and queryIdsWithFunding arrays respectively and will therefore show as funded as long as at least 1 token is sent. However, 1 token will in most tokens not be enough to cover the reward setup in the feed.

Therefore, users may submit data thinking they will get the feed base reward, but when the balance of the feed is lower than the base reward, they will get the balance of the feed which may be as small as 1 token.

Moreover, this difference in tokens the user receives is not accumulated, meaning once the user claims his tips he will get whatever the balance is, and he will not be able to recover the remaining amount until the reward in future claims.

Code Location:

**Listing 4: src/autopay/Autopay.sol (Line 189)**

```
160 function claimTip(
161     bytes32 _feedId,
162     bytes32 _queryId,
163     uint256[] calldata _timestamps
164 ) external {
165     Feed storage _feed = dataFeed[_queryId][_feedId];
166     uint256 _balance = _feed.details.balance;
167     require(_balance > 0, "no funds available for this feed");
168     uint256 _cumulativeReward;
169     for (uint256 _i = 0; _i < _timestamps.length; _i++) {
170         require(
```

```
171             block.timestamp - _timestamps[_i] > 12 hours,
172             "buffer time has not passed"
173         );
174         require(
175             getReporterByTimestamp(_queryId, _timestamps[_i]) ==
 ↳ msg.sender,
176             "message sender not reporter for given queryId and
 ↳ timestamp"
177         );
178         _cumulativeReward += _getRewardAmount(
179             _feedId,
180             _queryId,
181             _timestamps[_i]
182         );
183         if (_cumulativeReward >= _balance) {
184             // Balance runs out
185             require(
186                 _i == _timestamps.length - 1,
187                 "insufficient balance for all submitted timestamps
 ↳ "
188             );
189             _cumulativeReward = _balance;
190             // Adjust currently funded feeds
191             if (feedsWithFunding.length > 1) {
192                 uint256 _idx = _feed.details.feedsWithFundingIndex
 ↳  - 1;
193                 // Replace unfunded feed in array with last
 ↳ element
194                 feedsWithFunding[_idx] = feedsWithFunding[
195                     feedsWithFunding.length - 1
196                 ];
197                 bytes32 _feedIdLastFunded = feedsWithFunding[_idx
 ↳ ];
198                 bytes32 _queryIdLastFunded = queryIdFromDataFeedId
 ↳ [
199                     _feedIdLastFunded
200                 ];
201                 dataFeed[_queryIdLastFunded][_feedIdLastFunded]
202                     .details
203                     .feedsWithFundingIndex = _idx + 1;
204             }
205             feedsWithFunding.pop();
206             _feed.details.feedsWithFundingIndex = 0;
207         }
```

```
208             _feed.rewardClaimed[_timestamps[_i]] = true;
209         }
210         _feed.details.balance -= _cumulativeReward;
211         require(token.transfer(msg.sender, _cumulativeReward));
212         emit TipClaimed(_feedId, _queryId, _cumulativeReward, msg.
 ↳ sender);
213 }
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (5.0)**

Recommendation:

Ensure feeds / queries only show as feedsWithFunding and queryIdsWithFunding when they have at least enough balance to at least cover the base rewards.

Remediation Plan:

**RISK ACCEPTED**: The Fetch Oracle team accepted the risk of this issue as they considered that this issue is mitigated by external scripts being run in conjunction with Telliot. These scripts scan for profitable tips before submitting a price to Fetch Oracle. This allows reporters to only report when there are enough tips available for a profit.

## 4.4 (HAL-04) MISSING STORAGE GAPS - MEDIUM (5.2)

Description:

The contract UsingFetchUpgradeable is meant to be inherited by other contracts. However, it does not contain a __gap variable, even though it is upgradeable.

This means that in the event of adding new state variables when upgrading the contract, it can lead to storage slot collisions.

BVSS:

**AO:A/AC:L/AX:H/C:N/I:C/A:C/D:N/Y:N/R:N/S:C (5.2)**

Recommendation:

Consider adding a __gap variable in the UsingFetch contract.

Remediation Plan:

**SOLVED**: The Fetch Oracle team solved the issue in the following commit:

608339b7fe47043852b29474fbd2ef2412ba470b

## 4.5 (HAL-05) CHAINLINK HELPER CAN RETURN INCORRECT PRICES - MEDIUM (6.7)

The ChainlinkHelper contract retrieves the price from a Chainlink price feed. However, the updatedAt value is not being checked to check that the round is complete.

This can lead to the oracle returning an incorrect price, as the round might be incomplete.

Code Location:

**Listing 5: src/SecondaryOracle/ChainLinkHelper.sol**

```
17 function getPrice(bytes memory _data) external returns (bool,
↳ uint256, uint256, bool) {
18     (
19         /* uint80 roundID */,
20         int price,
21         /*uint startedAt*/,
22         uint timestamp,
23         /*uint80 answeredInRound*/
24     ) = aggregator.latestRoundData();
25
26     return (true, uint(price), timestamp, true);
27 }
```

BVSS:

**AO:A/AC:L/AX:M/C:N/I:C/A:N/D:N/Y:N/R:N/S:U (6.7)**

Recommendation:

Check the updatedAt parameter when getting the price.

**Listing 6: src/SecondaryOracle/ChainlinkHelper.sol**

```
require(updatedAt > 0, "Round is not complete");
require(answer >= 0, "Malfunction"); price");
```

Remediation Plan:

**SOLVED**: The Fetch Oracle team solved this issue since the Chainlink oracle price feed is not available in PulseChain, the Chainlink Helper was never intended to be used. Instead, the Secondary Oracle TWAP Helper is used.

## 4.6 (HAL-06) IMPLEMENTATION CONTRACT CAN BE INTIIALIZED - LOW (2.5)

Description:

There is no constructor in the FetchFlex, Governance, Autopay and SecondaryOracle contracts which disables initializers for the given contract.

This means that when deploying the implementation contract it is left uninitialized and available for anyone to initialize and become the owner of the contract, which is often a vector used in Phishing attacks.

BVSS:

**AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)**

Recommendation:

Implement a constructor calling the _disableInitializers() function.

Remediation Plan:

**SOLVED**: The Fetch Oracle team fixed the issue in the following commits:

- f7ae06d1728b5665c5e35f5edf4ec6c52eeb2a8a
- 5ac3f75e51cbb392825c5dd1d039bd7d02a5f97b
- 49f6f4022250b972fcf419c5c0f55e5457b74401
- a853d3770cafd19a55fbee74e6b83b20e7db2471

## 4.7 (HAL-07) REDUNDANT DATA OVERWRITING - LOW (2.5)

Description:

The storeData() function from the QueryDataStorage library, used in the Autopay contract. Stores the data of the query data related to its query ID in a mapping.

However, this function is called several times within the code even if the data is already stored, which leads to redundancy and unnecessary gas usage.

Code Location:

**Listing 7: src/autopay/Autopay.sol (Line 305)**

```
304 queryIdFromDataFeedId[_feedId] = _queryId;
305 queryDataStorage.storeData(_queryData);
306 emit NewDataFeed(_queryId, _feedId, _queryData, msg.sender);
307 if (_amount > 0) {
308     fundFeed(_feedId, _queryId, _amount);
309 }
310 return _feedId;
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:Uğ1‘ (2.5)**

Recommendation:

Only call the storeData function the first time a query data needs to be stored.

Remediation Plan:

**RISK ACCEPTED**: The Fetch Oracle team accepted the risk of this finding.

## 4.8 (HAL-08) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation:

Set the pragma to a fixed version.

Remediation Plan:

**ACKNOWLEDGED**: The Fetch Oracle team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN