CHAINLINK

Solidity Security Review

ConversionRate Smart Contract

Version: 2.0

May, 2019

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the `ConversionRate` smart contract, part of the ChainLink platform. This review focused solely on the security aspects of the Solidity implementation of the contract, but also includes general recommendations and informational comments relating to minimizing gas usage.

Sigma Prime performed a security assessment of the base ChainLink smart contracts in a previous engagement (Q4 2018).

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the contract (`ConversionRate`) contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/-closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as "informational". Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities found within the `ConversionRate` contract.

Overview

The ChainLink contracts can be logically separated into two parts:

- The Oracle entity (encapsulated in `Oracle.sol`);
- The ChainLink framework contracts (contained in `Chainlink.sol`, `Chainlinked.sol` and `ChainlinkClient.sol`).

An Oracle is an entity in the platform that fulfils requests for off-chain data. Contracts on the platform auction `LINK` tokens to request this data. The tokens are transferred to the Oracle contract, which emits an event to alert the Oracle a new request has been made. The contract to which the Oracle should send its off-chain data when fulfilling a request may cancel the request if the Oracle has been unresponsive after a pre-defined period of time. The `owner` of the Oracle contract can fulfil the request and claim the `LINK` tokens.

The ChainLink framework contracts provide a set of Solidity helper functions which users of the platform can inherit in their contracts. These functions help construct and correctly encode requests to get sent to the user's chosen Oracle.

Additionnally, the only smart contract in scope for this review, `ConversionRate.sol`, is an aggregation contract allowing users to average out the results from multiple oracles to generate a single value that can be considered more accurate (as it is not dependent on a single entity).

Security Review Summary

This review was initially conducted on commit e1d9a3a, and targets exclusively the sole file `ConversionRate.sol`. This smart contract inherits the `ChainkLinkClient.sol` and OpenZeppelin's `Ownable.sol` smart contracts (both considered out of the scope of this assessment).

Retesting activities targeted commit 0464448.

The manual code-review section of the reports, focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focuses on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

The testing team identified a total of four (4) issues during this assessment, all of them classified as informational.

All vulnerabilities identified during this assessment have been addressed by the development team.

To support this review, the testing team used the following automated testing tools:

- Rattle: `https://github.com/trailofbits/rattle`
- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output from these automated tools have been omitted from this report, but are available upon request.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within ChainLink's `ConversionRate` smart contract. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contract, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team;
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk;
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
CLC-01	Rounding Errors Can Lead to Erroneous Aggregated Results	Informational	Resolved
CLC-02	Denial of Service Due to Lack of Validation	Informational	Resolved
CLC-03	Potential Front-Running of Responses from Malicious Oracles	Informational	Closed
CLC-04	Miscellaneous General Comments	Informational	Resolved

CLC-01	Rounding Errors Can Lead to Erroneous Aggregated Results
Asset	`ConversionRate.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `currentRate` public state variable contains the median of the responses received by oracles (gets updated by the `updateLatestAnswer()` function).

When the number of responses is even (i.e. when `responseLength % 2 == 0`), the `currentRate` is calculated as follows:

$$\text{currentRate} = \frac{(\text{median1} + \text{median2})}{2}$$

Where:

- `median1` is the result of a *quickselect* on the array of responses at `middleindex` (i.e. the number of responses divided by 2)
- `median2` is the result of a *quickselect* on the array of responses at `middleindex + 1`

Since *Solidity* only deals with integers, the resulting rate is rounded down (An actual median of 3.5 would returned as 3). This issue is only informative as the typical way to handle such rounding errors is to scale the value to a desired precision via changing the representation of the number being calculated.

More clearly, if the value being returned is the wei rate of ETH, for example, and one would like to avoid a rounding error at the wei level, the representation being sent back by Oracles could be designated as $kilo - wei$. If a rate of ETH was 100 wei, then the number $100,000$ would be returned. The rounding error would then be at a $milli - wei$ precision.

Recommendations

This issue is raised as informative point only. The authors may be aware of this limitation and/or be satisfied with the current precision.

Higher precisions can be obtained via the representation of the returned value as explained above.

Resolution

The development team acknowledges the issue, see below response:

"Off-chain, the oracle job specification will include a task to multiply the API response by 108̂ (102̂ for cents, and then 106̂ for an additional 6 decimal places of precision).""

CLC-02	Denial of Service Due to Lack of Validation
Asset	`ConversionRate.sol`
Status	**Resolved:** In commit c4cfac5
Rating	Informational

Description

The `ConversionRate` smart contracts takes an unbound array of addresses `oracles` in its constructor (and when updating the Oracle list) to store addresses of the ChainLink nodes allowed to submit answers to requesters.

Due to the Ethereum block gas limit restriction, the `ConversionRate` smart contract becomes unusable if the `oracles` array contains too many addresses. Specifically, calling `requestRateUpdate()` can become prohibitively expensive.

Furthermore, Oracle's have a minimum limit of gas in which they must send back in the callback function. The current hard coded limit is $400,000$ gas. Thus there is a limit if the callback function exceeds this gas limit. The gas requirements of the callback function is dependent on the number of Oracles in the contract.

Testing revealed that:

- The contract cannot be deployed with an initial set of `oracles` greater than **142** oracles;
- The `requestRateUpdate()` function needs more than $8,000,000$ gas (current block gas limit) when the `oracles` array includes more than **75** oracles;
- The `updateRequestDetails()` function needs more than $8,000,000$ gas (current block gas limit) when the `oracles` array includes more than **189** oracles;
- If there are more than **49** oracles in the contract, the transactions sent from oracles that send the minimum gas ($400,000$) will fail silently (Oracle transactions will succeed but the `ConversionRate` contact will not be updated);
- Callbacks from unsorted response array require more gas (the *quickselect* function uses more gas for an unsorted array). In this instance, more than **45** oracles will cause the callback function to revert for Oracles that send the minimum amount of gas.

Effectively, the `ConversionRate` smart contract becomes unusable when the `oracles` array contains more than **75** oracles. The callback will revert for some Oracles (dependent on the gas they send) if more than **45** are used in the `ConversionRate` contract. Please refer to the supporting test suite for illustration `test_conversion_rate_requests.py`.

Recommendations

To prevent some of these potential issues for users of the contract, we recommend adding an additional validation requirement to ensure that the number of Oracles referenced by the `ConversionRate` contract is strictly less than **75** (and probably less than **45**).

Resolution

A limit on the number of oracles has been introduced by the development as per the recommendation above.

CLC-03	Potential Front-Running of Responses from Malicious Oracles
Asset	`ConversionRate.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `ConversionRate` contract updates a variable `currentRate` based on input from a number of oracles. Front-running [3] is possible as users are able to see relevant transactions from known oracles and can therefore perform front-running attacks based on the updated value of `currentRate`.

There also exists a mitigated attack vector where oracle's may try to front-run each other in order to bias `currentRate`. As the current rate is based on the median of the all oracle responses, this attack vector is minimized as 50% of chosen oracle's would need to collude to arbitrary set `currentRate`.

We raise this issue as informational, to inform the authors that the `currentRate` is able to be biased and this can become a significant risk if the variable is used in a critical manner, such as price of an item on a decentralized exchange. In this example, a user could buy and sell an item before and after the price updates via front-running the oracle's responses.

Recommendations

We raise this such that the authors are aware of the potential bias and possible front-running of the `currentRate` variable. This should be factored into applications that rely on this specific variable.

To prevent front-running amongst oracles who wish to bias the variable, a 2-round *commit-reval* scheme could deter oracles from front-running as they will not definitively know the impact of their response in advance.

Resolution

The development team acknowledges this issue and plans on introducing an *offchain* threshold signature scheme (using Schnorr signatures) to mitigate the associated risk. More details will be shared with the public once this scheme is finalised (see related PR.)

CLC-04	Miscellaneous General Comments
Asset	`ConversionRate.sol`
Status	**Resolved:** See inline comments
Rating	Informational

Description

This section describes general observations made by the testing team during this assessment, including gas savings suggestions.

1. **Public function can be made External:** The following functions are declared with a `public` visibility while there are no internal calls made to them:
   - `requestRateUpdate()`
   - `chainlinkCallback()`
   - `setAuthorization()`
   - `destroy()`

   It would be possible to change their declaration to be `external` functions.
   ✓ Resolved in commit [0d88413]
2. `minimumResponses` **and** `maxResponses` **can be changed to** `uint128`**:** Due to the way the Solidity compiler manages the state storage layout, changing the type of the `Answer` struct's first two elements from `uint256` to `uint128` can result in gas savings. This is because both variables can be loaded from a single `SLOAD` opcode.
   ✓ Resolved in commit [b818347]
3. **Typographical error in comments:** There is a minor typo in the comments on lines [241] and [242] (*memroy*).
   ✓ Resolved in commit [c4cfac5]

Recommendations

We suggest:

1. Updating the functions listed above to use the `external` modifier;
2. Changing the type of the first two elements of the `Answer` struct to `uint128`;
3. Fixing the minor typographical error.

Resolution

These recommendations have been implemented, see inline comments above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `pytest` framework was used to perform these tests and the output is given below.

Please note that tests marked as `XFAILED` are expected to fail after the updates introduced by the development following the initial review.

```
test_chainlink_callback.py::test_chainlink_callback_random                    PASSED   [4%]
test_chainlink_callback.py::test_chainlink_callback_restricted_gas            XFAILED  [8%]
test_conversion_rate.py::test_link_transfer                                   PASSED   [12%]
test_conversion_rate.py::test_authorization                                   PASSED   [16%]
test_conversion_rate.py::test_destroy                                         PASSED   [20%]
test_conversion_rate.py::test_ownership                                       PASSED   [24%]
test_conversion_rate_edge_cases.py::test_median_result[responses0]            PASSED   [28%]
test_conversion_rate_edge_cases.py::test_median_result[responses1]            PASSED   [32%]
test_conversion_rate_edge_cases.py::test_median_result[responses2]            PASSED   [36%]
test_conversion_rate_edge_cases.py::test_median_result[responses3]            PASSED   [40%]
test_conversion_rate_edge_cases.py::test_median_result[responses4]            PASSED   [44%]
test_conversion_rate_edge_cases.py::test_median_result[responses5]            PASSED   [48%]
test_conversion_rate_edge_cases.py::test_median_result[responses6]            PASSED   [52%]
test_conversion_rate_edge_cases.py::test_median_result[responses7]            PASSED   [56%]
test_conversion_rate_edge_cases.py::test_async_oracle_callback                PASSED   [60%]
test_conversion_rate_edge_cases.py::test_async_oracle_callback_more_than_min  PASSED   [64%]
test_conversion_rate_edge_cases.py::test_async_oracle_callback_less_than_min  PASSED   [68%]
test_conversion_rate_edge_cases.py::test_async_oracle_multiple_requests       PASSED   [72%]
test_conversion_rate_requests.py::test_request_update                         PASSED   [76%]
test_conversion_rate_requests.py::test_update_request_details                 PASSED   [80%]
test_conversion_rate_requests.py::test_constructor_oracle_limit               XFAILED  [84%]
test_conversion_rate_requests.py::test_update_request_details_oracle_limit    PASSED   [88%]
test_conversion_rate_requests.py::test_request_rate_update_oracle_limit       XFAILED  [92%]
test_deploy.py::test_deploy                                                   PASSED   [96%]
test_deploy.py::test_full_conversion_rate_deploy                              PASSED   [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

	High	Medium	High	Critical
Impact	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].

[3] Sigma Prime. Solidity Security - Front Running. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html#race-conditions`. [Accessed 2018].

σ'