// Smart Contract Security Assessment

08.01.2024 - 08.16.2024

v2.1
RFX Exchange

HALBORN

v2.1 - RFX Exchange

Prepared by: HALBORN

Last Updated 09/04/2024

Date of Engagement by: August 1st, 2024 - August 16th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	1	2	0	1	0

TABLE OF CONTENTS

1. Introduction

The `Relative Finance team` engaged Halborn to conduct a security assessment on their smart contracts, beginning on August 01, 2024, and ending on August 16, 2024. The security assessment was scoped to the smart contracts inside their `rfx-contracts` GitHub repository, located at https://github.com/relative-finance/rfx-contracts. The engagement was around the changes being done in the following pull reques https://github.com/relative-finance/rfx-contracts/pull/41.

2. Assessment Summary

The team at Halborn was provided two weeks for the engagement and assigned one full-time security engineer to assess the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to achieve the following:

- Ensure that the system operates as intended.
- Identify potential security issues.
- Identify lack of best practices within the codebase.
- Identify systematic risks that may pose a threat in future releases.

In summary, Halborn identified some security issues that were successfully addressed by the `Relative Finance team`.

3. Test Approach And 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.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

4. RISK METHODOLOGY

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

- **10** - CRITICAL
- **9 - 8** - HIGH
- **7 - 6** - MEDIUM
- **5 - 4** - LOW
- **3 - 1** - VERY LOW AND INFORMATIONAL

5. SCOPE

FILES AND REPOSITORY

(a) Repository: rfx-contracts

(b) Assessed Commit ID: feb04bf

(c) Items in scope:

- ./contracts/adl/AdlUtils.sol
- ./contracts/config/Config.sol
- ./contracts/Timelock.sol
- ./contracts/data/DataStore.sol
- ./contracts/data/Keys.sol
- ./contracts/deposit/DepositVault.sol
- ./contracts/deposit/ExecuteDepositUtils.sol
- ./contracts/error/Errors.sol
- ./contracts/event/EventEmitter.sol
- ./contracts/exchange/AdlHandler.sol
- ./contracts/exchange/DepositHandler.sol
- ./contracts/exchange/LiquidationHandler.sol
- ./contracts/exchange/OrderHandler.sol
- ./contracts/exchange/WithdrawalHandler.sol
- ./contracts/fee/FeeHandler.sol
- ./contracts/fee/FeeUtils.sol
- ./contracts/market/MarketFactory.sol
- ./contracts/market/MarketToken.sol
- ./contracts/oracle/IOracleProviderPayable.sol
- ./contracts/oracle/Oracle.sol
- ./contracts/oracle/OracleModule.sol
- ./contracts/oracle/OracleStore.sol
- ./contracts/oracle/OracleUtils.sol
- ./contracts/oracle/PythDataStreamProvider.sol
- ./contracts/position/DecreasePositionCollateralUtils.sol
- ./contracts/position/IncreasePositionUtils.sol
- ./contracts/pricing/PositionPricingUtils.sol
- ./contracts/role/Role.sol
- ./contracts/role/RoleModule.sol
- ./contracts/role/RoleStore.sol
- ./contracts/router/ExchangeRouter.sol
- ./contracts/router/Router.sol
- ./contracts/router/SubaccountRouter.sol
- ./contracts/swap/SwapHandler.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 53b871e

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	0	1	0

IMPACT X LIKELIHOOD

				HAL-01
				HAL-03 HAL-04
	HAL-02			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISMATCH BETWEEN STORED AND REPORTED PYTH PRICES	CRITICAL	SOLVED - 08/12/2024
MISSING PAYABLE KEYWORD RENDERS SOME FUNCTIONS USELESS	HIGH	SOLVED - 08/12/2024
NOT SENDING VALUE RENDERS WITHORACLEPRICESFORATOMICACTION MODIFIER USELESS	HIGH	SOLVED - 08/14/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PYTH ORACLE PRICE IS NOT VALIDATED PROPERLY	LOW	SOLVED - 08/12/2024

7. FINDINGS & TECH DETAILS

7.1 MISMATCH BETWEEN STORED AND REPORTED PYTH PRICES

// CRITICAL

Description

When the price of a given asset falls below `1`, Pyth oracle returns a positive `price` field **BUT** a negative `expo`. However, the contract `PythDataStreamProvider` retrieves such a value from the `dataStore`' s storage, being type-casted as an `uint256`. That makes the contract handle the price wrongly as for an exponent of `-1`, it will treat the price as if it had an exponent of `1`, which is completely erroneous as it does not reflect the actual value of the given asset.

Proof of Concept

The bug is pretty visual:
https://github.com/relative-finance/rfx-contracts/blob/feb04bf331c97a379556ed2eba30972415980034/contracts/oracle/PythDataStreamProvider.sol#L74

```
        uint256 precision = _getDataStreamMultiplier(token);
        uint256 adjustedBidPrice = Precision.mulDiv(uint256(minPrice), precision,
Precision.FLOAT_PRECISION);
        uint256 adjustedAskPrice = Precision.mulDiv(uint256(maxPrice), precision,
Precision.FLOAT_PRECISION);
```

The function being called is defined as:
https://github.com/relative-finance/rfx-contracts/blob/feb04bf331c97a379556ed2eba30972415980034/contracts/oracle/PythDataStreamProvider.sol#L87C1-L95C6

```
    function _getDataStreamMultiplier(address token) internal view returns (uint256) {
        uint256 multiplier = dataStore.getUint(Keys.dataStreamMultiplierKey(token));

        if (multiplier == 0) {
            revert Errors.EmptyDataStreamMultiplier(token);
        }

        return multiplier;
    }
```

It can be seen that the possible values for such a multiplier are only positive or zero, and the final price is multiplied by it. That makes it impossible to handle negative exponents that would otherwise return less-than-one prices.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:C/Y:N/R:N/S:C (10.0)

Recommendation

It is recommended to handle prices with negative coefficients by upcasting or downcasting the prices depending on the value of such a coefficient before doing any calculations.

Remediation Plan

SOLVED: The **Relative Finance team** solved this issue by updating the given `multiplier` depending on the value of the returned `expo` field from the Pyth price feed: https://github.com/relative-finance/rfx-contracts/blob/53b871efdaa63437c8397aa7bae4f3cdb6364ae5/contracts/oracle/PythDataStreamProvider.sol#L69C1-L87C117

```
            report.maxPrice = (r1.maxPrice * (10 ** 18)) / r2.minPrice;
            report.minPrice = (r1.minPrice * (10 ** 18)) / r2.maxPrice;
            report.expo = r1.expo - r2.expo - 18;

            report.publishTime = r1.publishTime > r2.publishTime ? r1.publishTime :
r2.publishTime;
        }

        if (report.minPrice <= 0 || report.maxPrice <= 0) {
            revert Errors.InvalidDataStreamPrices(token, int192(report.minPrice),
int192(report.maxPrice));
        }

        uint256 precision = _getDataStreamMultiplier(token);
        if (report.expo < 0) {
            precision = precision / (10 ** uint32(-1 * report.expo));
        } else {
            precision = precision * (10 ** uint32(report.expo));
        }
        uint256 adjustedBidPrice = Precision.mulDiv(uint256(report.minPrice), precision,
Precision.FLOAT_PRECISION);
        uint256 adjustedAskPrice = Precision.mulDiv(uint256(report.maxPrice), precision,
Precision.FLOAT_PRECISION);
```

Remediation Hash

https://github.com/relative-finance/rfx-contracts/commit/53b871efdaa63437c8397aa7bae4f3cdb6364ae5

7.2 MISSING PAYABLE KEYWORD RENDERS SOME FUNCTIONS USELESS

// HIGH

Description

Every call to `_validatePrices` should come from a `payable` function as it sends a fee to the Pyth provider **BUT** the functions `validatePrices` and `setPricesForAtomicAction` do not have such a keyword, as oposed to `setPrices`, so they render unusable. The place where ETH is needed is here.

Proof of Concept

The affected functions are:
https://github.com/relative-finance/rfx-contracts/blob/0466b4a880d7d5b87ff75b639358fbe53db3b313/contracts/oracle/Oracle.sol#L179C1-L184C6

```
    function validatePrices(
        OracleUtils.SetPricesParams memory params,
        bool forAtomicAction
    ) external onlyController returns (OracleUtils.ValidatedPrice[] memory) {
        return _validatePrices(params, forAtomicAction);
    }
```

https://github.com/relative-finance/rfx-contracts/blob/0466b4a880d7d5b87ff75b639358fbe53db3b313/contracts/oracle/Oracle.sol#L119C1-L125C6

```
    function setPricesForAtomicAction(
        OracleUtils.SetPricesParams memory params
    ) external onlyController {
        OracleUtils.ValidatedPrice[] memory prices = _validatePrices(params, true);

        _setPrices(prices);
    }
```

it can be seen they do **NOT** have the `payable` keyword, so calls to these functions can **NOT** carry ETH. However, they both call `_validatePrices`, which eventually calls:
https://github.com/relative-finance/rfx-contracts/blob/0466b4a880d7d5b87ff75b639358fbe53db3b313/contracts/oracle/Oracle.sol#L294

```
            OracleUtils.ValidatedPrice memory validatedPrice =
IOracleProviderPayable(provider).getOraclePrice{value: fee}(
                token,
                data
            );
```

but no ETH can be attached, so no fee can be sent to the Pyth provider, which reverts the transaction in those situation, rendering this core functionality useless.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:C/Y:N/R:N/S:C (10.0)

Recommendation

It is recommended to add the `payable` keyword to those functions that deal with `msg.value`, so that sending them ETH does not revert and can behave correctly.

Remediation Plan

SOLVED: The **Relative Finance team** solved this issue by adding the `payable` keyword in the affected functions: https://github.com/relative-finance/rfx-contracts/blob/53b871efdaa63437c8397aa7bae4f3cdb6364ae5/contracts/oracle/Oracle.sol#L179C1-L184C6

```
    function validatePrices(
        OracleUtils.SetPricesParams memory params,
        bool forAtomicAction
    ) external payable onlyController returns (OracleUtils.ValidatedPrice[] memory) {
        return _validatePrices(params, forAtomicAction);
    }
```

https://github.com/relative-finance/rfx-contracts/blob/53b871efdaa63437c8397aa7bae4f3cdb6364ae5/contracts/oracle/Oracle.sol#L119C1-L125C6

```
    function setPricesForAtomicAction(
        OracleUtils.SetPricesParams memory params
    ) external payable onlyController {
        OracleUtils.ValidatedPrice[] memory prices = _validatePrices(params, true);

        _setPrices(prices);
    }
```

Remediation Hash

https://github.com/relative-finance/rfx-contracts/commit/53b871efdaa63437c8397aa7bae4f3cdb6364ae5

7.3 NOT SENDING VALUE RENDERS WITHORACLEPRICESFORATOMICACTION MODIFIER USELESS

// HIGH

Description

Inside the `OracleModule` contract, `withOraclePricesForAtomicAction` modifier, there is a call to `setPricesForAtomicAction` inside the `Oracle` contract, which calls `_validatePrices` and so it sends a given fee to the Pyth provider for its services. However, no ETH is sent (without taking into account the previous issue around the `payable` keyword), so the Pyth endpoint will revert the transaction as no fee was provided.

Proof of Concept

Pretty visual:
https://github.com/relative-finance/rfx-contracts/blob/53b871efdaa63437c8397aa7bae4f3cdb6364ae5/contracts/oracle/OracleModule.sol#L35C1-L41C6

```
    modifier withOraclePricesForAtomicAction(
        OracleUtils.SetPricesParams memory params
    ) {
        oracle.setPricesForAtomicAction(params);
        _;
        oracle.clearAllPrices();
    }
```

As seen before, the call to `setPricesForAtomicAction` needs some ETH to be used as fees, but no ETH is sent, so any function with this modifier, namely `WithdrawalHandler::executeAtomicWithdrawal` renders useless.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:C/Y:N/R:N/S:C (10.0)

Recommendation

It is recommended to send the required fee attached to the call to `oracle.setPricesForAtomicAction`, so that the ETH is sent through the contract to the Pyth endpoint.

Remediation Plan

SOLVED: The **Relative Finance team** solved this issue by sending the required fee alongside the call to the `oracle`: https://github.com/relative-finance/rfx-contracts/pull/47/files#diff-ec9855fdb537cbb2d7605899ff9818f4988dbd02b59ccd34f148e721d5c0bc3c

```
    modifier withOraclePricesForAtomicAction(
        OracleUtils.SetPricesParams memory params
    ) {
        oracle.setPricesForAtomicAction{value: msg.value}(params);
        _;
        oracle.clearAllPrices();
    }
```

7.4 PYTH ORACLE PRICE IS NOT VALIDATED PROPERLY

// LOW

Description

The `PythDataStreamProvider` contract does not perform input validation on the `price`, `conf`, and `expo` values returned from the called price feed, which can lead to the contract accepting invalid or untrusted prices. Those values should be checked as clearly stated in the official documentation.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:L/Y:N/R:N/S:C (4.7)

Recommendation

The contract should revert the transaction here if one of the following conditions is triggered:

- `price <= 0`
- `expo < -18`
- `conf > 0 && (price / int64(conf) < MIN_CONF_RATIO` for a given `MIN_CONF_RATIO`

Remediation Plan

SOLVED: The **Relative Finance team** solved this issue by checking the recommended conditions, and reverting the transaction if the values were not the expected ones.

Remediation Hash

https://github.com/relative-finance/rfx-contracts/commit/53b871efdaa63437c8397aa7bae4f3cdb6364ae5

References

https://solodit.xyz/issues/m-01-pyth-oracle-price-is-not-validated-properly-pashov-audit-group-none-nabla-markdown

8. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Overall, the reported issues were not mostly low/informational issues that did not pose a real threat to the system, so they were not considered to be part of this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.