DxDAO

Carrot KPI

Smart Contract Security Review

Version: 2.0

September, 2022

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Carrot KPI smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

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 Carrot KPI smart contracts 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 within the Carrot KPI smart contracts.

Overview

Carrot KPI is a token that takes collateral assets to "bet" on predefined KPIs (goals). The system relies on the `Reality.eth` infrastructure to define whether the KPIs are achieved or not. KPIs are translated into questions, while `Reality.eth` provides answers to those questions. Carrot KPI's oracle contracts (i.e., instances of `ManualRealityOracle` contract) are responsible for forwarding the answers to the corresponding KPI Token contract.

There are several possible outcomes based on the results forwarded by oracles:

- If an oracle returns a result lower than the lower bound or invalid:
  - If the KPI Token is set to `allOrNone`, then the collateral amount is the minimum payout.
    * This means the original amount minus minimum payout is recoverable by the creator.
    * The tokens become worthless (or less valuable) because the collateral amounts reduce to minimum payouts.
  - Otherwise, if no `allOrNone`, then the difference between the collateral amount and minimum payout, multiplied by each oracle's weight is the recoverable amount by the creator.
- If the oracle returns a result equal or higher than the higher bound, then all the collaterals belong to the KPI tokens holder.

- If the oracle returns an intermediate result (value between lower bound and higher bound), then the creator's recoverable collateral amount depends on how far it was from reaching the target, multiplied by the oracle weight.

A KPI Token is created through `KPITokensFactory`. The factory contract is supported by two other contracts, namely `KPITokensManager` and `OraclesManager`. Both contracts are used to manage KPI Token templates and oracle templates respectively. When a user wants to create a new KPI Token, they can select which token template and oracle template to use for the new token contract.

Security Assessment Summary

This review was conducted on the files hosted on the Carrot KPI repository and were assessed at commit 3bdba21.

Specifically, the files in scope are as follows:

- `KPITokensManager.sol`
- `KPITokensFactory.sol`
- `OraclesManager.sol`
- `ERC20KPIToken.sol`
- `ManualRealityOracle.sol`

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The fixes of the raised issues were re-assessed at commit 64d8a9b. Changes in this version are shown in the following table.

Old Contract Name	New Contract Name
ManualRealityOracle.sol	RealityV3Oracle.sol
OraclesManager.sol	OraclesManager1.sol
KPITokensManager.sol	KPITokensManager1.sol

In the new version, identical functions in `OraclesManager` and `KPITokensManager` were merged into a new contract, `BaseTemplatesManager`. The new contract was then derived by `OraclesManager1` and `KPITokensManager1`.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes 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 focused 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].

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

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

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 11 issues during this assessment. Categorized by their severity:

- Medium: 3 issues.
- Low: 1 issue.
- Informational: 7 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Carrot KPI smart contracts. 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 contracts, 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
CAR-01	Unsafe Typecasting During Oracle Finalization	Medium	Closed
CAR-02	Unredeemable Collateral Upon Token Expiry	Medium	Resolved
CAR-03	Incorrect Template Removal on `removeTemplate()`	Medium	Resolved
CAR-04	`unchecked` Arithmetic Operations	Low	Resolved
CAR-05	Ineffective Initialisation of `_kpiTokensManager` and `_oraclesManager`	Informational	Closed
CAR-06	No Check on `_oracleData.value`	Informational	Resolved
CAR-07	Potential Off-by-one Error on `enumerate()`	Informational	Closed
CAR-08	Changes to `KPITokensManager` May Not Propagate to `ERC20KPIToken`	Informational	Closed
CAR-09	Creator Approves Collateral Tokens to A Predicted Address	Informational	Closed
CAR-10	KPI Token Creator Influences Oracle Results	Informational	Closed
CAR-11	Miscellaneous General Comments	Informational	Resolved

CAR-01	Unsafe Typecasting During Oracle Finalization		
Asset	`ManualRealityOracle.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The `ManualRealityOracle` contract handles the finalisation of `KPIToken` results. The intention from the development team is to leverage the use of `Reality.eth` which is a stake-based oracle service aimed at answering arbitrary questions.

The function `finalize()` does not comply with the `Reality.eth` specifications. On line [**112**], function `finalize()` expects the result of `IReality(reality).resultFor(questionId)` to return a `uint256`, whereas `Reality.eth` returns type `bytes32`.

Therefore, there is a typecasting from `bytes32` to `uint256`. In certain situations, this may have negative repercussions. For instance, if a data provider responding to the oracle (i.e., calling `submitAnswer()` on `Reality.eth`) responds with `int256(-1)`, this will be stored as `bytes32(0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)`.

This may potentially have unintended outcomes leading to the release of collateral tokens when it should not.

The testing team believes that this edge-case could happen, given that users submitting answers to oracle questions may not have any knowledge of the internal workings of Carrot KPI contracts.

Recommendations

Ensure that the behaviour is understood and acknowledged. The testing team recommends avoiding unsafe typecasting and working with the returned type `bytes32` from `Reality.eth`.

Resolution

To address the issue, changes were made on commit a642398. The changes include upgrading to `RealityV3` protocol and limiting the number of `Reality.eth` template to four for a better visibility to the related parties, where manual intervention might be needed if an incorrect answer is submitted.

From the testing team's understanding of `Reality.eth`, there are no guarantees that safe type casting is possible as the protocol expects arbitrary answers to questions in the form of `byte32`. Even if arbitrators can correct answers, checks should be used to limit the risk of unsafe typecasting. For example, solutions that allow for safe typecasting may use bitwise operators to convert negative, zero and positive value answers to values of 0,1,2. Alternatively, if the development team desires flexibility, typecasting to a signed integer value may be more appropriate as this will not alter the value of an answer significantly (as in the case of `byte32 -> uint`).

CAR-02	Unredeemable Collateral Upon Token Expiry		
Asset	`ERC20KPIToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Function `redeemCollateral()` is callable by a KPI Token holder that has registered their redemption intent and burned their KPI Tokens through the function `registerRedemption()`. The function conducts several checks, including preventing the call when `_isExpired()` is `true`. This check is potentially incorrect. If the user has called `registerRedemption()` but does not call `redeemCollateral()` before the token expires, they will never be able to redeem their collateral tokens. Another indication that the check on line [**665**] is incorrect is that the function has a condition where it checks whether the token is expired or not on line [**675**].

Based on the current implementation, once the contract expires (i.e., when `expired() == True` holds), there is no way to finalise the contract.

Recommendations

The testing team recommends removing the instructions on line [**665**] to allow the burned token (through function `registerRedemption()`) to be redeemed.

Resolution

The issue has been fixed on commit 82b98e2. The code on line [**665**] was removed such that users can redeem their collateral tokens after the contract expires.

CAR-03	Incorrect Template Removal on `removeTemplate()`		
Asset	`KPITokensManager.sol, OraclesManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The `removeTemplate()` function of `KPITokensManager` contract removes a template from the `templates` array. The removed template is selected by the caller (owner) based on the template ID specified in the input parameter `_id`. To get the actual array index of the template, the function uses the `templateIdToIndex` mapping.

There is an issue in the implementation of this function. Specifically, if the removed template is not the last item in the array, the function does not delete the respective `templateIdToIndex`. As a result, if the user calls `removeTemplate()` again with the same template ID, the function will delete the wrong template.

Another related issue discovered during contract testing is that `templateToIndex` of the last template is assigned with an incorrect index on line [**176**], because the value of `_index` has been decremented by one on line [**170-172**] to facilitate `templates` index update on line [**175**]. As a result, the last template will be assigned with an incorrect index which compromises the correctness of `templateToIndex`.

Here is an example of what could go wrong. The user removes a template of id 0. The implementation correctly removes the template of id 0, but then assigns `templateIdToIndex[10]` to 0. As a result, the owner can not remove the template with id 10 because `templateIdToIndex[10]` returns 0 (on line [**168**]), which does not pass the check on line [**169**].

The same issue can be found in the `removeTemplate()` function of the `OraclesManager` contract.

Recommendations

For `KPITokensManager`, the bug can be remedied by deleting the template from the `templateIdToIndex` mapping on all occasions and fixing the `_index` value when assigning `templateIdToIndex`. The following snippet shows an example of how to mitigate this issue on line [**174-179**]:

```
174  if (_lastTemplate.id != _id) {
       templates[_index] = _lastTemplate;
176    templateIdToIndex[_lastTemplate.id] = _index + 1;
     }
178  delete templateIdToIndex[_id];
```

Alternatively, the function `removeTemplate()` can be rewritten as follows:

```
167 function removeTemplate(uint256 _id) external override onlyOwner {
      uint256 _index = templateIdToIndex[_id];
169   if (_index == 0) revert NonExistentTemplate();
      Template storage _lastTemplate = templates[templates.length - 1];
171   if (_lastTemplate.id != _id) {
        templates[_index-1] = _lastTemplate;
173     templateIdToIndex[_lastTemplate.id] = _index;
      }
175   delete templateIdToIndex[_id];
      templates.pop();
177   emit RemoveTemplate(_id);
    }
```

The testing team also recommends adding extensive tests to ensure the updates work as expected.

Resolution

The issue has been fixed on commit a9a7c1c. Function `removeTemplate()` was fixed following the recommendation.

CAR-04	`unchecked` Arithmetic Operations		
Asset	`ERC20KPIToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `calculateProtocolFee()` function conducts multiplication and division operations to an input value with constant values. These operations cannot be considered safe enough to use `unchecked` arithmetics as integer overflows can be triggered if the input value is too big or too small. With `unchecked`, these potential overflows will not be automatically detected.

As a result of wrapping unsafe arithmetic operations with `unchecked`, the function may not return an expected value. This behaviour can be mitigated by using the default checks available on the EVM.

Recommendations

The testing team recommends removing `unchecked` from function `calculateProtocolFee()`.

Resolution

The issue has been fixed on commit aedb0c9. The wrapper `unchecked` was removed from the related code as per recommendation.

CAR-05	Ineffective Initialisation of `_kpiTokensManager` and `_oraclesManager`
Asset	`KPITokensFactory.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

There is a circular reference between `KPITokensFactory`, `OraclesManager`, and `KPITokensManager` as follows:

- `KPITokensFactory` needs `KPITokensManager` and `OraclesManager` on `constructor`.
- `OraclesManager` needs `KPITokensFactory` on `initialize()`.
- `KPITokensManager` needs `KPITokensFactory` on `constructor`.

As a result, the deployment of `KPITokensFactory` requires the following workaround:

1. Deploy `KPITokensFactory` using dummy non-zero addresses for `OraclesManager` and `KPITokensManager`.
2. Use the factory address in the deployment process of `OraclesManager` and `KPITokensManager`.
3. Update the `OraclesManager` address and `KPITokensManager` address on `KPITokensFactory` using `KPITokensFactory.setOraclesManager()` and `KPITokensFactory.setKpiTokensManager()` respectively.

The structure above indicates inefficiency of the `KPITokensFactory`'s contract `constructor` because it involves the use of non-zero dummy addresses which should be replaced at later stages using the workaround as described above.

Recommendations

Taking dummy addresses on the constructor might be ineffective. The testing team recommends removing inputs `_kpiTokensManager` and `_oraclesManager` for the `constructor`, and also removing the related input checks on line [**40-42**]. Instead, an initialisation function can be implemented to assign the `kpiTokensManager` and `oraclesManager` variables, once they are available. This solution also requires additional checks on related functions, to ensure the `kpiTokensManager` and `oraclesManager` addresses on `KPITokensFactory` have been properly configured.

Resolution

The development team acknowledged and marked the issue as no fix with the following comment:

"I've decided to ignore this. It could be cause of concern and end up in unintended misconfiguration, but the deployment script should take care of it."

CAR-06	No Check on `_oracleData.value`
Asset	`ERC20KPIToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Creating and initialising a new `ERC20KPIToken` instance from `KPITokensFactory.createToken()` require several steps, including instantiating one or more `Reality.eth` oracles (or wrappers) through `OraclesManager` (for example, by using `ManualRealityOracle` template). This `Reality.eth` oracles instantiation involves payment (in native tokens i.e., ETH) to `Reality.eth` contract, that is, when calling `RealityETh.askQuestionWithMinBond()` as specified in `ManualRealityOracle.sol` on line [**90-94**].

On each `Reality.eth` oracle instantiation data, the payment to `Reality.eth` contract is specified in `_oracleData.value` (see line [**336**] of `ERC20KPIToken.sol`). However, there is no check to ensure that the sum of `_oracleData.value` is at least equal to `msg.value` of transaction `KPITokensFactory.createToken()`. If the sum of `_oracleData.value` is less than `msg.value`, then the transaction will fail and the gas spending could have been high before it fails.

On the other hand, if the contract creator sends too many native tokens, the surplus is unrecoverable and remains in `ERC20KPIToken`.

Recommendations

The testing team recommends checking the mentioned values as soon as possible to reduce the risk of the transaction failing that may have spent a lot of gas for contract creations and function calls. Such check may be implemented in function `initializeOracles()`.

To prevent a failing transaction caused by this issue, the development team could implement a helper in the UI to ensure that the KPI Token creator sends enough native tokens. The development team could also implement a function to allow the remaining native tokens to be returned to the transaction sender.

Resolution

The issue has been addressed on commit 64d8a9b. The development team decided to implement a check to ensure enough payment is sent with the transaction.

CAR-07	Potential Off-by-one Error on `enumerate()`
Asset	`KPITokensFactory.sol, KPITokensManager.sol, OraclesManager.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

In `KPITokensFactory` contract, the `enumerate()` function returns existing/created KPI Token contracts to the caller, where the returned items are sliced from the array `kpiTokens` by the given indices (from index or `_fromIndex` and to index or `_toIndex`). When `kpiTokens` has three items and the requested from and to indexes are 0 and 2, intuitively, the returned array must have three items (of index 0, 1, and 2). However, the current implementation returns two items, which indicates an *off-by-one* error.

This is because the iteration on line [**143**] starts from zero to `_range`, where `_range` is the difference between the `_toIndex` and `_fromIndex`. So, when `_toIndex` and `_fromIndex` are 0 and 2, then `_range` equals 2. The code on line [**143**] iterates only twice, because `_range` is an exclusive value.

The function returns three items when the inputs are 0 and 3. Instructions on line [**139**] also indicate that the maximum `_toIndex` is at most the length of the array `kpiTokens`. This means that the caller's `_toIndex` is an exclusive value, which is normal in a programming perspective, but counter-intuitive in an interface that requires user inputs.

The same issue also occurs on `KPITokensManager.enumerate()` and `OraclesManager.enumerate()`.

Recommendations

The testing team recommends updating the implementation to clarify the behaviour.

For `KPITokensFactory` contract, the testing team recommends adding 1 to `_range` on line [**141**], for example:

```
uint256 _range = _toIndex - _fromIndex + 1;
```

and modifying line [**139**] to:

```
if (_toIndex > (kpiTokens.length - 1) || _fromIndex > _toIndex)
```

Alternatively, if the current implementation is preferred than the suggested fixes above, add a user documentation to clearly indicate the function's required inputs to get the desired outputs.

Resolution

The development team acknowledged and marked the issue as no fix because it is a preferred behaviour. As a follow-up, the related documentation has been updated for clarity.

CAR-08	Changes to `KPITokensManager` May Not Propagate to `ERC20KPIToken`
Asset	`KPITokensFactory.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The contract `KPITokensFactory` allows arbitrary users to create KPI Tokens with configuration details set by a `KPITokensManager`. The `KPITokensManager` deploys a new KPI Token contract based on the user-chosen template ID (in function `KPITokensManager.instantiate()`).

Function `KPITokensFactory.setKpiTokensManager()` is only callable by its owner (enforced by modifier `onlyOwner`). This function can cause changes to the templates for new KPI Tokens. As there is no way to correct the `kpiTokenTemplate` within the `ERC20KPIToken` contract, any changes to the `kpiTokensManager` could cause unintended behaviour.

Consider the following scenario. A transaction that calls `setKpiTokensManager()` is placed in front of some other transactions that call `createToken()`. The new `KPITokensManager` have a completely different set of templates compared to the old `KPITokensManager`. This may create undesirable results, e.g., the created KPI Tokens were generated from different templates than what the creators intended.

Recommendations

Ensure that the behaviour is understood and acknowledged.

Resolution

The development team acknowledged and marked the issue as no fix.

CAR-09	Creator Approves Collateral Tokens to A Predicted Address
Asset	`ERC20KPIToken.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

When creating a new KPI Token, the contract creator must calculate the address of the to-be-created KPI Token through `KPITokensManager.predictInstanceAddress()`. In order to do so, the contract creator must have prepared all necessary data (i.e., input parameters to `KPITokensFactory.createToken()`). Then, after the predicted token address is calculated, the creator approves each collateral token to the predicted address, so when the new KPI Token is created, the collateralised amount of each token can be transferred to the KPI Token contract itself and the fee amount to the fee receiver.

The mechanism above has several potential issues:

- When the creator decides to modify the variables after approving tokens, existing approvals must be updated by removing the existing approvals then adding new ones.
- When the system variables change (i.e., preferred templates no longer exist), the predicted address will be different, which make the approvals void.

Recommendations

Make sure this behaviour is understood. The testing team suggests adding a vault contract to store and manage collateral tokens of the KPI Tokens. The vault can help simplify the contract creator's effort to fund the newly created KPI Token.

Alternatively, the development team could implement a separate function to allow the KPI Token contract creator to fund the collateralised amounts separately after the contract is created.

Resolution

The development team acknowledged and marked the issue as no fix with emphasis on future improvement.

CAR-10	KPI Token Creator Influences Oracle Results
Asset	`ManualRealityOracle.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The Carrot KPI protocol intends to utilise `Reality.eth` infrastructure to get information whether the KPIs set during the creation of a KPI Token is achieved. The KPI Token has one or more Reality oracle wrapper (e.g. `ManualRealityOracle` contract) to handle question submission and answer retrieval from `Reality.eth`.

There are two possible ways for the KPI Token creator to influence oracle results:

1. The creator supplies "tricksy questions" to the `Reality.eth`, such that the questions do not have clear answers or go to an arbitrator.
2. When creating a `ManualRealityOracle`, the creator must also supply an arbitrator address. According to Reality.eth documentation, the arbitrator will provide a final answer when a dispute about a question arises. The documentation specifies that `Reality.eth` allows the user to select the arbitrator: whether they prefer taking an option provided by the `Reality.eth` UI or supplying their own arbitrator.

   A malicious KPI Token creator might be tempted to use an arbitrator account they control. When the arbitrator is involved in a dispute, the final answer supplied by the arbitrator might be in the creator's favour, since the creator has an influence over the arbitrator account.

Recommendations

Make sure this behaviour is understood.

The testing team understands that `Reality.eth` is not within the scope of the review and the impact of the issue raised here might be higher than just Informational. However, this behaviour may have security implications affecting KPI Tokens.

Resolution

The development team acknowledged and marked the issue as no fix with emphasis on future improvement.

CAR-11	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Excessive Emitted Data on Event `Initialize()`**

   Related Asset(s): ERC20KPIToken.sol, ManualRealityOracle.sol

   Event `Initialize()` in contract `ERC20KPIToken` and `ManualRealityOracle` is emitted during initialisation phase of both contracts. This event emits several information, including initialisation data which can be gas intensive. Our test indicates that removing initialisation data from the emitted event reduces 27,568 gas or roughly 1.2% from the total gas spending.

   The testing team recommends removing the initialisation data from the emitted event. The gas cost can be further reduced by simplifying emitted data on other events, for example event `InitializeOracles`.

2. **Inefficient Function `protocolFee()`**

   Related Asset(s): ERC20KPIToken.sol

   The function `protocolFee()` is used to calculate the protocol fee for a KPI Token. The function takes an array of struct `TokenAmount` that contains token addresses and the amount for each token. Inside the function, no operation is done on the token addresses and instead, it is forwarded as a part of the encoded output. While the protocol fee of each collateral token is computed by calling `calculateProtocolFee()`.

   If `calculateProtocolFee()` is marked as `public`, a user can just call it to verify the protocol fee by specifying `_amount`, instead of calling `protocolFee()` that requires address-amount tuple.

   The testing team believes that a smart contract must maintain concise and efficient code for the purpose of achieving optimum performance and cost. The testing team recommends removing function `protocolFee()` and updating function `calculateProtocolFee()` from `internal` to `public`.

3. **Inefficient Function `finalizableOracle()`**

   Related Asset(s): ERC20KPIToken.sol

   Function `finalizableOracle()` searches through variable `finalizableOracles` to get a record of `FinalizableOracle` that matches the given oracle address. The operation cost of this function is `O(n)` where `n` is the number of oracles in `finalizableOracles`. This indicates that the gas cost will be more expensive on KPI Tokens with more oracles.

   The testing team recommends modifying `finalizableOracles`. Instead of using a standard `FinalizableOracle` struct array, it is advised to use a `mapping` to store the data, with oracle address poses as a mapping key. The following example shows how to do this:

   ```
   mapping(address => FinalizableOracle) internal finalizableOracles;
   ```

   Using the new variable above, the search cost can be reduced to `O(1)`. Then, the code in line [**365-373**] can be changed to:

```
365 FinalizableOracle storage _finalizableOracle = finalizableOracles[_address];
    if (
367 !_finalizableOracle.finalized &&
    _finalizableOracle.addrezz == _address
369 ) return _finalizableOracle;
    else revert Forbidden();
371
```

A further improvement can be made by removing `FinalizableOracle.addrezz` if the data is not used in any part of the system. Then, to check the existence of a `_finalizableOracle` record, one can use `if (_finalizableOracle.higherBound > 0) revert Forbidden();`

To complement the suggested changes above, a new variable that stores oracle addresses as index is useful for returning values in function `oracles()`.

4. **Typos**

   Related Asset(s): ERC20KPIToken.sol

   - line [**445**]: payour: payout
   - line [**460,463,470,473,514,517,573,576,577,578**]: _reimboursement : _reimbursement
   - line [**136**]: chosed: chosen

5. **Redundant Check on `finalize()`**

   Related Asset(s): ERC20KPIToken.sol

   Function `finalize()` checks whether the contract has been initialised, as indicated on line [**409**].

   This check is redundant because the function cannot work without initialisation. For example, if the caller is not one of the oracles in `finalizableOracle`, the function call will fail. This is because `finalizableOracle` will be assigned in `initializeOracles()` function, which is called by `initialize()` function.

   Therefore, calling `finalize()` before initialisation will revert with `Forbidden()` produced by the code on line [**411**].

   The code in line [**409**] can be safely removed.

6. **Inefficient Token Search on `collaterals`**

   Related Asset(s): ERC20KPIToken.sol

   Function `recoverERC20()` allows the `ERC20KPIToken` creator to recover any excess tokens sent to the contract. The function takes the token address and goes through the `collaterals` to check whether the token address is in the list (line [**551-553**]). This potentially wastes gas, for example if the token is not one of the collaterals.

   A similar issue can be found in function `redeemCollateral()` on line [**668-670**].

   The testing team recommends adding a variable that holds an index of the `collaterals`' addresses.

   Our test for `recoverERC20()` indicates **35%** reduction in gas spending when recovering non-collateral ERC20 tokens if codes on line [**551-572**] are deactivated.

7. **Inverted Variables When Emitting `RedeemCollateral`**

   Related Asset(s): ERC20KPIToken.sol

   Function `redeemCollateral()` emits an event `RedeemCollateral` when a collateral is successfully redeemed. The event is emitted with the following parameters:

   - `msg.sender`
   - `_token`
   - `_token`
   - `_receiver`
   - `_redeemableAmount`

While based on the specification, the event `RedeemCollateral` requires the following parameter sequence:

- `address indexed account`
- `address indexed receiver`
- `address collateral`
- `uint256 amount`

It indicates that the variable `_token` (on line [**694**]) and `_receiver` (on line [**695**]) are inverted.

The testing team recommends inverting the variables `_token` and `_receiver` in the event `RedeemCollateral` on line [**692-697**].

8. **Unused Error `Forbidden()`**

   Related Asset(s): KPITokensFactory.sol

   The custom error `Forbidden` in line [**24**] is not used in any part of the contract and can be safely removed.

9. **Unused Error `NoKeyForTemplate()`**

   Related Asset(s): KPITokensManager.sol, OraclesManager.sol

   The custom error `NoKeyForTemplate` is not used in any part of the contract and can be safely removed.

10. **Redundant Checks**

    Related Asset(s): ERC20KPIToken.sol

    Function `initializeState()` is called by function `initialize()` as a part of the initialisation process. The function assigns some input parameters to the contract's variables. Before doing so, this function conducts some checks; this includes checking whether the creator (variable `_creator`) and KPI Tokens Manager (variable `_kpiTokensManager`) are not zero address (on line [**213-214**]). These checks are redundant if it is always assumed that `ERC20KPIToken` is properly deployed through `KPITokensFactory.createToken()`. This is because:

    - `msg.sender` is the creator of the contract (`_creator`) which cannot be a zero address (so far, the private key for the zero address is not known)
    - `_kpiTokensManager` is taken from `KPITokensFactory`'s variable, `kpiTokensManager`, which has been sanitised during contract constructor.

    Therefore, the checks in line [**213-214**] can be safely removed.

    Similarly, ensuring that `_oraclesManager` is not a zero address in function `initializeOracles()` is redundant because `KPITokensFactory.oraclesManager` is guaranteed to be a non-zero address. Therefore, the check in line [**315**] can be safely removed.

11. **Inefficient Collateral Check on `collectCollateralsAndFees()`**

    Related Asset(s): ERC20KPIToken.sol

    The codes on line [**268-270**] of function `collectCollateralsAndFees()` check whether there are any duplicates in the list of collaterals. This is inefficient because it involves a linear search where each element is checked against all other elements. In a list with five elements, this will require four iterations with a total of 4+3+2+1 = 10 comparisons with 10 data access to `collaterals`.

    The testing team recommends modifying the function logic to improve the efficiency of the check. A mapping that points to the index of the collaterals is recommended.

12. **`MULTIPLIER` Changes Values on Excessively High `amount`**

    Related Asset(s): ERC20KPIToken.sol

    Function `handleLowOrInvalidResult()` and `handleIntermediateOrOverHigherBoundResult()` utilise `MULTIPLIER` in calculating reimbursement amount for the KPI Token owner. In a case where `_collateral.amount` has an excessively high value (e.g., 2^240), left-shifting `_numerator` with `MULTIPLIER` will cause an accuracy loss on

`_numerator`. As the result, the value of `_reimboursement` will be lower compared to operations without the multiplier operations (left-shift then right-shift).

Make sure this behaviour is understood.

13. **Input `bytes` Instead of `address` in Function `redeem()`**

    Related Asset(s): ERC20KPIToken.sol

    Function `redeem()` can be used by the KPI Token holders to redeem the collateral tokens. The function takes an input parameter `_data` which determines the destination address of the collateral tokens. The input parameter `_data` is of type `bytes` instead of `address`. The code on line [**602**] decodes the input parameter `_data` to get the destination address.

    The testing team discovered that changing `bytes` to `address` does not have significant impact on gas improvement (600 gas reduction when using `address`). However, taking `address` instead of `bytes` as an input would make function `redeem()` clearer for the users to understand the purpose of the input parameter. It is also worth noting that the suggested change requires an adjustment to `IKPIToken.sol`.

14. **Gas Optimisation on `redeem()`**

    Related Asset(s): ERC20KPIToken.sol

    Function `redeem()` emits an event `Redeem` when collaterals are successfully redeemed. The event is emitted with `_redeemedCollaterals` (that contains the list of collaterals and their amounts) as a part of the emitted data. This set of data might be too excessive, as it takes at least 4.6% (11,649 gas) of the transaction's gas usage. If a user wants to get the details of what collaterals are redeemed and the amounts, they can find the information on `Transfer` events emitted by function `safeTransfer()`.

    The testing team recommends removing `_redeemedCollaterals` to reduce the gas usage.

15. **Remaining Balances After Redeem**

    Related Asset(s): ERC20KPIToken.sol

    Contract `ERC20KPIToken` may have remaining collateral token balances after all KPI Tokens are redeemed/burned. This is because of accumulated rounding errors that may occur during token redemption. The amounts, however, are insignificant.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The responses to the raised issues are as follows.

1. Fixed on commit 252f0df. The event `Initialize` now does not contain any parameters.
2. Changes were made on commit 252f0df to improve gas consumption.
3. Fixed on commit 8ba10aa. A mapping is added to optimise search.
4. Fixed on commit 719cd06.
5. Fixed on commit 0986d2e. The redundant check was removed.
6. Fixed on commit 8ba10aa. The array `collaterals` was no longer used and replaced with mappings.

7. Fixed on commit 4390f9f.
8. Fixed on commit 54cd637. The unused `Error` was removed.
9. Fixed on commit d1197f7. The unused `Error` was removed.
10. Fixed on commit 1d94651. The redundant checks were removed.
11. No changes were made.
12. No changes were made.
13. No changes were made.
14. Fixed on commit 259bc5f. Variable `_redeemedCollaterals` was removed from event `Redeem`.
15. No changes were made.

Appendix A Test Suite

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

```
test_init                                          PASSED   [1%]
test_protocol_fee                                  PASSED   [3%]
test_finalize_invalid_all_or_none                  PASSED   [5%]
test_finalize_invalid_not_all_or_none              PASSED   [7%]
test_finalize_low_all_or_none                      PASSED   [9%]
test_finalize_intermediate_all_or_none             PASSED   [10%]
test_finalize_over_all_or_none                     PASSED   [12%]
test_finalize_invalid_over_all_or_none             PASSED   [14%]
test_finalize_invalid_intermediate_all_or_none     PASSED   [16%]
test_finalize_invalid_intermediate_not_all_or_none PASSED   [18%]
test_finalize_intermediate_invalid_all_or_none     PASSED   [20%]
test_finalize_not_initialized                      PASSED   [21%]
test_recover_ERC20                                 PASSED   [23%]
test_recover_ERC20_expired                         PASSED   [25%]
test_redeem_expired                                PASSED   [27%]
test_redeem_expired_check_balance                  SKIPPED  [29%]
test_redeem_finalised_check_balance                SKIPPED  [30%]
test_register_redemption                           PASSED   [32%]
test_register_redemption_finalize                  PASSED   [34%]
test_redeem_collateral_expired                     PASSED   [36%]
test_redeem_collateral_finalized                   PASSED   [38%]
test_create_token                                  SKIPPED  [40%]
test_create_token                                  SKIPPED  [41%]
test_duplicate_oracle                              SKIPPED  [43%]
test_create_token_usdt                             SKIPPED  [45%]
test_setup_protocol                                PASSED   [47%]
test_setup_protocol_configure                      PASSED   [49%]
test_constructor                                   PASSED   [50%]
test_create_token                                  PASSED   [52%]
test_enumerate                                     XFAIL    (Intendedbehav...)
test_enumerate_invalid_indices                     PASSED   [56%]
test_set_kpi_tokens_manager                        PASSED   [58%]
test_set_oracles_manager                           PASSED   [60%]
test_set_fee_receiver                              PASSED   [61%]
test_init                                          PASSED   [63%]
test_constructor                                   PASSED   [65%]
test_add_template                                  PASSED   [67%]
test_remove_template                               PASSED   [69%]
test_upgrade_template                              PASSED   [70%]
test_update_template_specification                 PASSED   [72%]
test_enumerate                                     PASSED   [74%]
test_finalize_invalid                              PASSED   [76%]
test_finalize_intermediate                         PASSED   [78%]
test_finalize_higher                               PASSED   [80%]
test_data                                          PASSED   [81%]
test_template                                      PASSED   [83%]
test_computeMul                                    SKIPPED  [85%]
test_init                                          PASSED   [87%]
test_initialize                                    PASSED   [89%]
test_instantiate_fails                             PASSED   [90%]
test_add_template                                  PASSED   [92%]
test_remove_template                               PASSED   [94%]
test_upgrade_template                              PASSED   [96%]
test_update_template_specification                 PASSED   [98%]
test_enumerate                                     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.

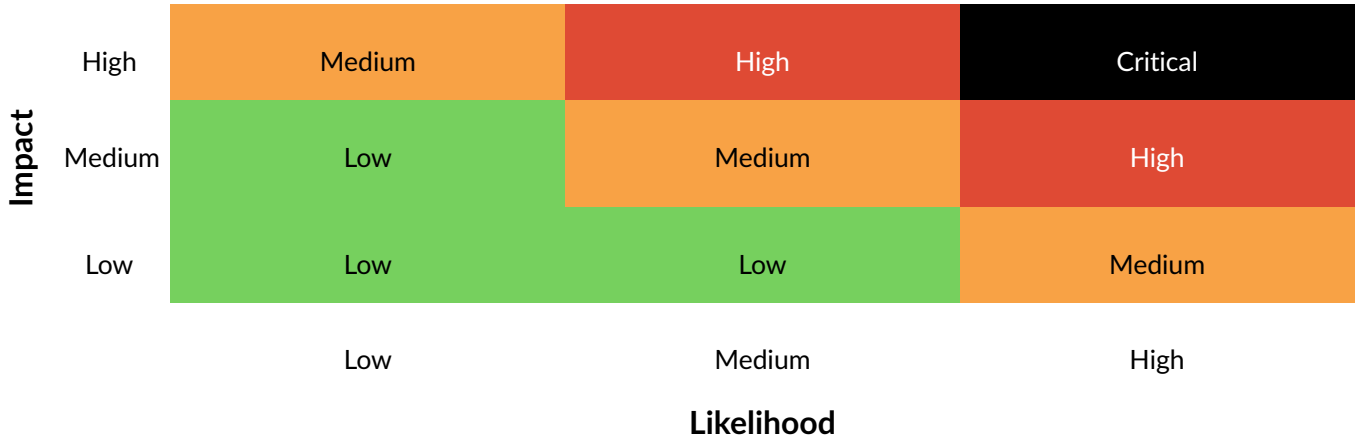

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].