DXdao Carrot

Final Audit Report

July 18th, 2022

Team Omega

teamomega.eth.limo

Summary

DXdao has asked Team Omega to audit the contracts that define the behavior of the Curve wrapper for Swapr contracts.

We found **1 high severity issue** - this is an issue that can lead to a loss of funds, and is essential to fix. We classified **1** issue as "medium" - this is an issue we believe you should definitely address. In addition, **20** issues were classified as "low", and **13** issues were classified as "info" - we believe the code would improve if these issues were addressed as well.

These issues were subsequently addressed by the DXdao team, and Omega reviewed the fixes.

Severity	**Number of issues**	**Number of resolved issues**
High	1	0
Medium	2	1
Low	18	16
Info	14	12

Scope of the Audit

We audited the code from the following repository and commit:

```
      https://github.com/carrot-kpi/contracts-v1/commit/203c1595880edbd65b5efee
71cb8bcd808832138
```

The audit concerns all Solidity code under the `contracts` folder of the repository.

Resolution

DXdao subsequently addressed the issues in commit

```
      82b98e232fd490fc1b993c3e0dc2695c03a58d26
```

We reviewed these fixes and updated the issues below.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the variables and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

Centralization Risks

The Carrot contracts define a set of external administrative roles that can change certain aspects of the system and that will be held by external accounts (either personal accounts or multisig wallets).

We identify the following roles and the risk associated with these roles

The **KPITokensFactory.owner** can call:

```
setKpiTokensManager
setOraclesManager
setFeeReceiver
```

The **KPITokensManager.owner** can call

`addTemplate`
`removeTemplate`
`upgradeTemplate` (and `updateTemplateSpecification`)

The **OraclesManager.owner** can call

`addTemplate`
`removeTemplate`
`upgradeTemplate` (and `updateTemplateSpecification`)

The **ERC20KPIToken.creator** gets all the KPI tokens and can call

`recoverERC20` and recover any surplus ERC20 tokens from the contract

The rights of the three owner roles are all directly related to managing (creating, updating and deleting) the contracts that are available to creators of Carrot tokens, and as such are limited in scope and specifically, should only influence the options available to the token creators. However, as we pointed out in issue F1, the account that holds these roles has way more power than merely setting the parameters within which a Carrot token creator can act: we believe this constitutes a security risk.

General

G1. Incomplete test coverage [info] [not resolved]

The project uses Foundry as its development framework. Foundry does not have a test coverage reporting tool (yet), so we do not have any precise statistical information on test coverage. We do know, however, that coverage is not complete:

- `ERC20KIPToken.template()` is not tested at all
- The `NoKeyForTemplate` error in `OraclesManager.removeTemplate` is not tested
- The `NoKeyForTemplate` error in `KPITokensManager.removeTemplate` is not tested

Recommendation: Improve the test coverage of the code.
Severity: Info
Resolution: The issue was not resolved

G2. Follow Checks-Effects-Interaction pattern where possible [info] [resolved]

It is best practice to do, when possible, any state changes before external calls. This mitigates the risk of re-entrancy bugs. The pattern is not always followed:

- `ManualRealityOracle.finalize()`
- `ERC20KPIToken.initialize()`
- `ERC20KPIToken.initializeOracles()`
- `ERC20KPIToken.collectProtocolFees()`
- `ERC20KPIToken.redeem()`

We found no concrete exploit. However, `initializeOracle` is vulnerable in case the Oracle Manager address is compromised - but in that case there could be worse exploits than reentrancy.
Recommendation: Follow the Check-Effects-Interaction pattern, even where the `nonReentrant` modifier is used.
Severity: Info
Resolution: This issue was resolved as recommended.

G3. Event parameters are not indexed [info] [resolved]

None of the event parameters in the contracts are currently indexed. This limits the ways that the event log can be searched directly by event parameters as filters, which would make utilizing the events in the UI more challenging.
Recommendation: Index event parameters where appropriate.
Severity: Info
Resolution: This issue was resolved as recommended.

G4. Use solidity 0.8.15 [low] [resolved]

Solidity version 0.8.14, which is used in the audited contracts, contains two "important bugs" that were fixed in 0.8.15 - https://github.com/ethereum/solidity/releases/tag/v0.8.15 . These bugs are not directly relevant to the audited code, but we recommend upgrading the version in any case.
Recommendation: Upgrade the solidity version
Severity: Low
Resolution: This issue was resolved as recommended.

ERC20KPIToken

E1. A single collateral token could block redeeming of all collateral of a token owner [medium] [resolved]

The `redeem` function attempts to redeem a user's KPI tokens for collateral tokens by calling the `transfer` function of each token in a loop. If the `transfer` of one token fails (for example due to it being malicious, or paused), none of the collateral tokens will be redeemable.
Recommendation: Ensure collateral tokens can be withdrawn individually. This could be done by separating the redeeming of KPI tokens from the withdrawal of collaterals, for example by saving a balance of the redeemer after burning their tokens, then allowing them to withdraw tokens from that balance separately. This could be in addition to the currently available option, as the current path may be more convenient for users, but it should exist to ensure redeeming cannot be prevented due to a single token.
Severity: Medium
Resolution: This issue was resolved as recommended. Collaterals can now be withdrawn individually.

E2. Malicious or non-responsive oracle could lock tokens forever [medium] [partially resolved]

Collateral tokens become available for the contract creator once an oracle is finalized, and for the token holders only after all oracles are finalized (or if one finalized as failed in an and relationship configuration).
This means that if an oracle fails to finalize for any reason (malicious oracle, inactive oracle controller, bugs in oracle code etc.), some or all the collateral will be locked in the token contract forever. In case the oracle is malicious, this could also be used to extort the token creator and/ or the token holders up to the amount in risk to be locked in the token contract.
Recommendation: Add an expiration date for the `KPIToken` after which all oracles that were not finalized can be marked as failed, and collaterals will be distributed accordingly.
Severity: Medium
Resolution: This issue was partially resolved. An expiration date was added, so collaterals will not be stuck in the contract even if an oracle fails to finalize.
However, with the current logic, if a single oracle fails to finalize before the expiration date, KPI token holders will only get a minimum payout. In other words, the output of the oracles that did finalize correctly are ignored (also if allOrNothing is false). We believe this behavior is inconsistent - KPI tokens are now worth more if an oracle reports a negative or invalid result compared to when an oracle fails to respond at all. It also creates a security risk, as it may offer a malicious KPI token creator or an outsider that shorts the KPI token a path to “cheat” the token holders even if certain goals are reached, and some of the oracles are successfully finalized.

E3. Initialize oracles and collect fee lack access restriction [low] [resolved]

The functions `initializeOracles` and `collectProtocolFees` have no access restrictions, and can be called by any account. These functions will typically be called by the Factory contract as part of the deployment process, and in this context, the lack of access restrictions poses no security risk, as they can be called only once during the lifetime of the contract.. However, to be safe for future or independent uses and to save gas by reducing the amount of external calls necessary to initialize the token, we recommend that you fix the issue by making these functions part of the `initialize` function.
Recommendation: Mark the functions `initializeOracles` and `collectProtocolFees` as `internal` and call them as part of the initialization process, inside the `initialize` function.
Severity: Low
Resolution: This issue was resolved as recommended.

E4. Use constant for constant values [low] [resolved]

The following state variables are constant, and can be marked as such to save some gas on deployment:

```
uint256 internal immutable INVALID_ANSWER =
    0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
uint256 internal immutable MULTIPLIER = 64;
```

Recommendation: Mark these variables as `constant` instead of `immutable` to save some gas.
Severity: Low
Resolution: This issue was resolved as recommended.

E5. Remove unnecessary struct [low] [resolved]

In the initialize function, arguments are copied into `struct InitializeArguments`. This serves no particular purpose, and some gas can be saved by removing this wrapping into a struct, and deleting the struct itself.
Recommendation: Remove the struct and its use from the code to save some gas.
Severity: Low
Resolution: This issue was resolved as recommended.

E6. ReentrancyGuard is improperly initialized [low] [resolved]

The contract is inheriting from the Open Zeppelin `ReentrancyGuard` contract, which initialized its state inside a constructor. But since the contract is meant to be used in a proxy pattern, the constructor is not called for every instance deployed, and the `ReentrancyGuard` will be improperly initialized.
Recommendation: Inherit the `ReentrancyGuardUpgradeable` instead and call the `__ReentrancyGuard_init()` function inside the `initialize` function to properly initialize the state of the `ReentracyGuard` of each deployed proxy.
Severity: Low - the `ReentrancyGuard` logic will work also without the initialization code.
Resolution: This issue was resolved as recommended.

E7. Finalized flag is unnecessary [low] [resolved]

The `toBeFinalized` variable will necessarily always be equal to 0 when and only when the contract is finalized. This means the `finalized` variable could be removed from the contract - the `toBeFinalized` variable already indicates that the contract is finalized when it is equal to 0.
Recommendation: Remove the `finalized` variable.
Severity: Low
Resolution: This issue was resolved as recommended.

E8. Duplicated check that the oracle is not finalized [low] [resolved]

On line 339 there is a check of the contract is finalized or the oracle is finalized, but whenever the contract is finalized, all oracles are set to finalized too, and on line 338, the `finalizableOracle` call already checks that the specific token is not finalized, which means the check is practically duplicated and could be removed.
Recommendation: Remove the check on line 339.
Severity: Low
Resolution: This issue was resolved as recommended.

E9. Wrong final oracle progress set when result is bigger than the oracle higher bound [low] [resolved]

On line 378, if the final result is higher than the oracle higher bound, the value of `_oracle.finalProgress` is set to the oracle higher bound. This means that if for example, the oracle lower bound was 20 and the higher bound 30, and the oracle final result returned was 31. The correct final progress would be 11 - not 30.
Recommendation: Remove the condition on the final oracle progress value setting and always assign its value to be the oracle result minus the lower bound. Or if for any reason you would like to avoid the final progress being more than the maximum, set it to the oracle full range instead of the higher bound on line 379.
Severity: Low
Resolution: This issue was resolved as recommended. Now if an oracle has failed to reach the minimum, its `finalResult` will be 0, and if it went over the maximum, it will be the value of the full range (and if in between, it will be the actual progress made).

E10. template() function might not return the template used to initialize the current instance [low] [resolved]

The template function queries the KPITokenManager with the `templateId` used when initializing the oracle. The documentation of the function states that:

```
returning info about the template used to instantiate this KPI token.
```

This is not always the case. The oracle manager can update the template(and change all its properties) or remove it, and so the `template` function may return any information whatsoever - or fail in case the template was removed.
Recommendation: There are different ways to resolve this, depending on your use case. If you need this information to be available in the UI only, remove this function and rely on events; if instead this information is needed on-chain, store the template information at initialization, and use that information. Consider also F1 below. .
Severity: Low. It is not clear if this function will be used for any critical functionality, but at best, this function will lead to confusion.
Resolution: This issue was resolved as recommended. The template is now saved upon initialization so its value is not influenced by changes in the KPITokenManager.

E11. Remove unused code [info] [resolved]

The following errors are not used can can be removed:

```
error InconsistentWeights();
error InconsistentCollaterals();
error NoFunding();
error InconsistentArrayLengths();
```

Recommendation: Remove the unused errors from the code.
Severity: Info
Resolution: This issue was resolved as recommended.

E12. Variables saved but never read could be removed [info] [not resolved]

The description and the oracles’ final progress are saved in the contract, but are never read and could be just outputted in events instead of wasting gas on saving them.

Recommendation: Remove the description variable and the oracle final progress variable and instead just have their values posted in events.
Severity: Info
Resolution: This issue was not resolved.

KPITokensFactory

F1. The owners of KPITokensManager, KPITokensFactory can take all the collaterals of a newly deployed token [high] [not resolved]

The `KPITokensManager` manages a set of implementation contracts for tokens. Such implementation contracts can be added, removed or upgraded by the owner.
Upgrades are done by calling `upgradeTemplate`, which changes properties of the template, and updates the version property of the contract, but leaves the id of the contract unchanged.
Templates are accessed by their Id - i.e. it is not possible to address the template by its version.

A more serious problem is that it makes the result of calling `createToken` not completely predictable - the caller of the function can not be sure which token template will be used for the deployment, because the template associated with the ids that she provided as arguments can change at any time. Specifically, the owner of the `KPITokensManager` contract can front-run the transaction in which `createToken` is called, update the token template and replace it with a malicious contract, and gain access to all collateral sent to this new token contract.

A very similar attack can be executed by the owner of the KPITokensFactory, which can frontrun by calling `setKpiTokensManager` and change the token manager with a malicious contract.

Finally, the owner of the `OracleManager` can, in a slightly more complex attack, deploy malicious Oracle contract templates. Although we see no way for her to directly obtain funds, she will effectively control how the KPIToken is finalized, which means she controls the release of the funds, and potentially can extort nearly up to the amount of the collaterals in the token to allow releasing them.
Recommendation: Remove the versioning logic completely, and make templates uniquely addressable by their ID (i.e. calling `template(id)`) with a specific ID always returning the exact same template information.
Severity: High - all collateral funds provided by the KPI Token’s creator are at risk, and the size of the attack (*all* collateral is at risk) means that successfully bribing or otherwise compromising a quorum of multisig signers becomes a real possibility.
Resolution: This issue was not resolved.

F2. Remove unused imports [info] [resolved]

The contract imports the `Clones` contract from Open Zeppelin, as well as the `IOraclesManager` interface, but neither is used in the contract.
Recommendation: Remove the unused imports.
Severity: Info
Resolution: This issue was resolved as recommended.

KPITokensManager

KM1. Salt function is prone to collisions [low] [resolved]

The salt function used to predict the address of new instances is calculated based on parameters that are not necessarily unique for a caller.
This means that it is possible for a third party to deploy a new token with the same parameters and create a token contract in the same address.
In the current implementation of the contacts - i.e. in the way that the `KPITokensFactory` and `ERC20KPIToken` are intended to be used - this could be used as a griefing attack, where an attacker front-runs a `createToken` call, then call with the same parameters to "steal" the instance, causing the original deployer's transaction to fail. This would not be a very fruitful attack as the owner can tweak parameters and retry, while the attacker would have to supply the collateral each time.
Another type of risk is when users of the contracts send assets or approvals to that address in the expectation that the correct KPI contracts will be deployed to the pre-calculated address. This can be a problem, as has been seen recently with Gnosis Safe (https://mirror.xyz/banteg.eth/iZAsBNL3j_5NIAY2Erav1r7Q4ecc7SC76AfMjyScs34). Of course, approving the collateral tokens before actual deployment is the pattern used in the current factory, and we see no immediate risk here; but the current contracts are intended to be used as an extendable framework, there is a risk that future implementation will be less careful.
Recommendation: Include the original deployer address in the salt calculation, just like done in the Oracles Manager's `salt` function.
Severity: Low
Resolution: This issue was resolved as recommended.

KM2. Use top-level variables instead of a struct [low] [resolved]

The contract has a state variable `templates` that is a `struct` wrapping variables that could themselves be top-level.

This `struct` is not used for any purpose. However, it makes reading variables more costly, and it limits the visibility of the individual variables.
Recommendation: Remove the EnumerableTemplateSet from the code and have its variables as top level variables of the contract with appropriate visibility assigned to each.
Severity: Low
Resolution: This issue was resolved as recommended.

KM3. Template exists variable is unnecessary [low] [resolved]

The `Template` struct contains a boolean variable `exists`, which is used to signify whether a template exists for a given ID.
This variable can be removed, as there are other ways of determining if a template exists or not. For example, if template does not exist, its ID is 0, which cannot be so for templates that do exist (if deleted properly in case `removeTemplate` is called, see issue KM4) as IDs start from 1.
This issue applies to the `OraclesManager` as well.
Recommendation: Remove the `exists` variable from the template struct and verify that the template ID is not 0 where check of existence is needed.
Severity: Low
Resolution: This issue was resolved as recommended.

KM4. Properly delete template data after removal [low] [resolved]

The `removeTemplate` function only sets the removed template's `exists` variable to false, but keeps all the template data. It would save more gas to delete the whole data, which the contract no longer needs.
This issue applies to the `OraclesManager` as well.
Recommendation: Delete the entire template data when calling `removeTemplate`.
Severity: Low
Resolution: This issue was resolved as recommended.

KM5. Unbounded loop on removing a template [low] [resolved]

The `removeTemplate` function contains an unbounded loop on the keys array. If there are too many items, the loop might run out of gas, causing the function to fail and the removal of templates at the end of the array impossible, at least not without removing other templates.
This issue applies to the OraclesManager as well.
Recommendation: For a quick fix, allow the caller to specify the ID location in the array, which will make the loop obsolete. For a better solution, consider using the pattern used in OpenZeppelings

`EnumerableSet` - i.e to store the Templates in an array (instead of in a key map) and keep a map from ids to indexes in the array for accessing Templates by their id.
Severity: Low
Resolution: This issue was resolved as recommended.

KM6. Enumerate might return unexpected templates [info] [resolved]

The `enumerate` function is expected to return a list of the templates sliced based on indexes. To do that it uses the keys array, assuming that it contains an ordered list of the IDs of the templates. However, this is not necessarily the case. The keys array is disorganized when removeTemplate is called with an ID that is not the latest in the keys array. For example, if the enumerate function is asked to return for indexes 1 to 3 out of 5 templates, after template 2 was deleted, it will return [1, 5, 3] while it would be expected to return [1, 3, 4].
This issue applies to the OraclesManager as well.
Recommendation: The `enumerate` function does not appear to be useful, and we would recommend removing it completely. In case there is a strong reason to keep it, we recommend that you at least include this possible misordering in the function documentation.
Severity: Info
Resolution: This issue was resolved. The function documentation was updated to reflect the possible misordering of the templates.

KM7. Duplicated check template exists on remove template [info] [resolved]

The `removeTemplate` function calls the `storageTemplate` function, which verifies that a template exists for a certain ID and then returns it, or reverts if it doesn't exist. Then at the end of the `removeTemplate` function, there is a revert that is meant to be triggered in case the template was not found in the loop (i.e. if the template does not exist). This would be impossible, since the call to `storageTemplate` already checked for existence.
This issue applies to the OraclesManager as well.
Recommendation: Remove the `revert` statement on line 165.
Severity: Info
Resolution: This issue was resolved as recommended.

M8. Factory address should be marked as immutable [info] [resolved]

The factory address never changes and could be marked immutable to save gas.
Recommendation: Mark the `factory` variable as immutable.

Severity: info
Resolution: This issue was resolved as recommended.

KM9. Use the onlyOwner modifier [info] [resolved]

The contract inherits from Open Zeppelin's `Ownable` contract, but does not use its `onlyOwner` modifier. Instead, the functions which need to check that verify the caller is the owner use duplicated code for the check, which is against the DRY principle.
Recommendation: Use the `onlyOwner` or create your own modifier and use it instead.
Severity: Info
Resolution: This issue was resolved as recommended.

OraclesManager

OM1. Significant code duplication across the manager contacts [low] [not resolved]

A large part of the code in `KPITokenManager` and `OraclesManager` (and their interfaces) is duplicated. This specifically holds for the template management code, which is, apart from minor differences, the same.
Recommendation: The contracts could be refactored into a single manager contract, which could be deployed twice, with one managing KPITokens and the other oracles. Alternatively, in case you do believe certain differences are inevitable, the duplicated code could be extracted into a separate contract, from which both managers could inherit.
Severity: Low - but be aware that the issues KM3, KM4, KM5, KM6 and KM7 equally apply to the OraclesManager contract.
Resolution: This issue was not resolved.

OM2. Unused parameter "_automateable" [low] [resolved]

The add template accepts an `_automatable` parameters, which it then stores as part of the template data in the template struct. This variable is only allowed to be set to false - trying to set it to true will revert with an `AutomationNotSupported` error.. Thus, there is no reason for it to exist in the contract and it should be removed.
Recommendation: Remove the unused `_automatable` parameter and its related code.
Severity: Low
Resolution: This issue was resolved. The contract was updated to be upgradeable, which will allow future versions to utilize the `_automateable` flag.

OM3. Missing check template address is not 0 on upgrade template [info] [resolved]

The `upgradeTemplate` function is missing the requirement that the new template address must not be the 0 address, as is required on adding a template.
Recommendation: Add a check to verify the new template address is not 0.
Severity: Info
Resolution: This issue was resolved as recommended.

ManualRealityOracle.sol

M1. template() function might not return the template used to initialize the current instance [low] [resolved]

The template function queries the OracleManager with the templateId used when initializing the oracle. The documentation of the function states that:

```
returning info about the template used to instantiate this oracle.
```

This is not always the case. The oracle manager can update the template with a new version (and change all its properties), and so the template function may return a template of a newer version of the ID (which does not correspond to the one used in the contract), or even fail in case the template was removed from the oracle manager.
Recommendation: There are different ways to resolve this, depending on your use case. If you need this information to be available only in the UI, remove this function and rely on events; if instead this information is needed on-chain, store the template information at initialization, and use that information.
Severity: Low. It is not clear if this function will be used for any critical functionality, but at best, this will lead to confusion.
Resolution: This issue was resolved as recommended. The template is now saved upon initialization so its value is not influenced by changes in the OracleManager.

M2. Reality.eth parameter is misnamed [info] [resolved]

The contract takes at initialization a parameter called expiry, which is documented as:

```
/// - `uint32 _expiry`: The question expiry as described in the
Reality.eth docs (linked above).
```

Yet no such parameter exists in reality.eth. The corresponding parameter is called `opening_ts` and is documented as:

```
The opening_ts is the timestamp for the earliest time at which it will be
possible to post an answer to the question. You can use 0 if you intend
the question to be answerable immediately.
```

Naming it as `expiry` in the code might cause a confusion as it doesn't properly reflect neither the original parameter name nor its actual function.
Recommendation: Rename the `expiry` parameter in the code to `opening_ts` or any name that would correspond better to its actual use.
Severity: Info
Resolution: This issue was resolved as recommended. The variable was renamed to `_openingTimestamp`.

M3. Unnecessary check that question is finalized [low] [resolved]

On line 109 the `require` statement verifies that the question is finalized on reality.eth. This check is already done by the reality.eth contract when calling the `resultFor` function on line 112, so this check can be removed
Recommendation: Remove the check to save gas.
Severity: Low
Resolution: This issue was resolved as recommended.

M4. Consider to remove the Oracle contract once finalized [info] [not resolved]

After an oracle contract has been finalized successfully, it has no more use or reason to exist. Therefore, calling `selfdestruct` and removing the contract of the blockchain at the end of the `finalize` function would clean up the unnecessary component, saving some gas and reducing possible attack surface.
Recommendation: Instead of managing the oracle's finalization state with a variable, call self destruct after the oracle was successfully finalized. Note that it is crucial to ensure the self-destruct cannot be called on the original contract, which can be done by disabling calls to finalize in the constructor.
Severity: Info
Resolution: This issue was not resolved.

M5. Oracle is limited in arbitrator selection and bond configuration [info] [resolved]

The `initialize` function asks a question to the reality.eth oracle. It takes a `_data` argument, parses that, and passes the resulting data on to the reality.eth's `askQuestion` function. However, two parameters can not be set with the current logic:

- The `min_bond` parameter is always set to 0.
- No ether is sent along with the `askQuestion` call, which means that:
    - If the arbitrator requires a fee, the transaction will fail.
    - No bounty for answering the question can be added (although a bounty can be added later).

Recommendation: Allow specifying the `min_bond` and sending ETH to the oracle when asking the question to take full advantage of the features offered by the oracle.
Severity: Info
Resolution: This issue was resolved as recommended.