OpenZeppelin | security

Polymarket UMA Optimistic Oracle Adapter Audit

August 16, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-07-17 To 2023-07-28
Languages	Solidity

Total Issues	12 (9 resolved, 3 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	3 (1 resolved, 2 partially resolved)
Low Severity Issues	3 (2 resolved, 1 partially resolved)
Notes & Additional Information	5 (5 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the `Polymarket/uma-ctf-adapter` repository at the 11647f1 commit.

In scope were the following contracts:

```
src
├── UmaCtfAdapter.sol
├── mixins
|   ├── Auth.sol
|   └── BulletinBoard.sol
└── libraries
    ├── AncillaryDataLib.sol
    └── TransferHelper.sol
```

System Overview

Polymarket is a prediction market on the Polygon blockchain. The protocol allows for the creation of markets where users can buy and sell positions based on the perceived likelihood of a future event happening, such as the results of an election. A key component that is necessary for determining the outcome of a market is a trusted oracle. The contracts in scope enable Polymarket to interface with UMA's Optimistic Oracle protocol, such that questions can be created and resolved via the Oracle. Below is an overview of the `UmaCtfAdapter`, the main contract in the system.

`UmaCtfAdapter` Contract

The `UmaCtfAdapter` contract can function both as a wrapper around Gnosis's Conditional Tokens Framework and as an oracle to the `NegRiskOperator` contract. Users are able to `initialize` questions which will then be instantiated as price requests on UMA's Optimistic Oracle. When the Optimistic Oracle successfully resolves a question, users will then be able to resolve the question in the adapter and report the payout of the result to either the Conditional Token Framework or the `NegRiskOperator`.

The `UmaCtfAdapter` contract is also capable of handling disputes on the Optimistic Oracle by enabling the callback feature in the Oracle. If a dispute occurs, the `priceDisputed` function will be called, at which point a new price request will be created on the Optimistic Oracle where the adapter will track that price request moving forward. This was done to minimize the time delay when disputes occur, as it was observed by Polymarket that most outcomes are disputed once. Therefore, forcing a user to initiate more than one dispute discourages the practice unless there is a systemic issue with the question. Upon a second dispute within the Optimistic Oracle for a given question, Polymarket will wait for that dispute to settle.

As a safety mechanism, the `UmaCtfAdapter` has administrative functions that allow a privileged address to flag, pause/unpause, force a new price request for a question, and emergency-resolve a result for a question that has been flagged.

Security Model and Trust Assumptions

- Admins are expected to not be malicious. They will promptly flag questions with potentially incorrect/indeterminate outcomes and honestly emergency-resolve them. Also, it is expected that there will always be at least one admin since the `Auth` contract allows the removal of all admins.
- The Optimistic Oracle and the `UmaCtfAdapter` are capable of supporting price requests that end in ties or are otherwise unable to have their result determined. However, when integrating with the `NegRiskOperator`, ties are not supported. Therefore, it is expected that the `UmaCtfAdapter` should only create questions that do not result in ties/indeterminate outcomes.
- None of the contracts in the system are upgradeable. This reduces the level of trust required in the contract deployers, as once the contracts are deployed, they cannot be changed. In the event of a vulnerability in the system, it could not be fixed.
- The `UmaCtfContract` allows the ancillary data to be specified in a separate contract that allows for changes to the data. It is usually expected by the Optimistic Oracle that the ancillary data passed in the price request corresponds to the question and cannot change. Therefore, it is assumed that the proposers of results to the Optimistic Oracle will correctly observe and respond to price requests whose question data may change.

Privileged Roles

`UmaCtfAdapter`:

- Admins can:
  - Flag a question, thus preparing the question to be emergency-resolved after a time delay.
  - Reset a question, which creates a new price request to the Optimistic Oracle.
  - Emergency-resolve a question, which ignores the resolved price request determined by the Optimistic Oracle.
  - Pause/unpause a question in order to temporarily block question resolution.

High Severity

H-01 Reward Tokens Can Get Stuck in `UmaCtfAdapter`

The creator of a question can specify a `rewardToken` and the `reward` amount they want to give to the resolver of an UMA Optimistic Oracle price request.

As a result of the `callbackOnPriceDisputed` flag being set to `true`, if anyone disputes a price request, the reward tokens are sent back to the `UmaCtfAdapter`, and `priceDisputed` is called. This resets the question, sending the reward tokens back to the Optimistic Oracle. If the question is disputed a second time, the reward tokens are sent back to the `UmaCtfAdapter` but the callback will not reset the question anymore. This creates the possibility that the reward tokens remain stuck in the adapter unless a user calls `resolve` and the price returned by the UMA Optimistic Oracle is the `ignorePrice`, which is unlikely to occur often.

Consider tracking the flow of reward tokens, and assessing on a case-by-case basis whether the tokens need to be refunded to the creator of the question or kept in order to fund a user resetting the question.

***Update**: Resolved in pull request #61 at commit 866fcc5.*

Medium Severity

M-01 Missing Documentation on Correctly Initializing Questions

The `reward`, `proposalBond` and `liveness` parameters of `UmaCtfAdapter`'s `initialize` function should be carefully chosen based on the particularities of each question.

UMA's documentation advises on how to reason about these parameters:

- Based on how much value can be moved by a question, different `proposalBonds` might need to be set to ensure both proposers and disputers are incentivized to participate in the oracle process.
- Based on the type of markets and questions, different `liveness` might need to be set.

Consider documenting the internal team's reasoning about the values used during initialization, and writing guidelines on how to set these based on the questions encountered so far.

***Update**: Resolved in pull request #62 at commit 736cdee.*

M-02 `UmaCtfAdapter` Cannot Report Ties When Used as the Oracle to the `NegRiskOperator`

As part of the UMIP for `YES-OR-NO` queries, the specification requires a value to be defined in the event that an answer to a question cannot be determined. In the event of a tie or indeterminate result for a question, the resulting payout when created via the `_constructPayouts` helper function will be `[1, 1]`.

When the `UmaCtfAdapter` is used alone, this is an acceptable outcome that can be successfully reported to the Conditional Token Framework. However, when the `UmaCtfAdapter` is used as the oracle to the `NegRiskOperator`, this outcome is incompatible. This is because the `reportPayouts` function in the `NegRiskOperator` will revert when the passed-in payouts array is `[1, 1]`. Therefore, when the `UmaCtfAdapter` goes to report a payout indicating a tie, the call will revert. This will prevent a question from resolving, and thus prevents the market that the question is a part of from being able to redeem users' positions.

While the intent is to never ask a question that could result in a tie, it is still possible for this situation to arise in the future. Therefore, consider clearly documenting this risk so that users can understand this edge case. Also consider documenting the verbiage and procedures for creating a question. This can help ensure consistency across questions and reduces the likelihood of accidentally instantiating a question whose result may end in a tie or be indeterminate.

***Update**: Partially resolved in pull request #71 at commit a1abb68. The added comment should ideally contain more details on what would happen if the result is a tie.*

M-03 Interactions Between the `UmaCtfAdapter` and the `NegRiskOperator` Are Not Documented

The `NegRiskOperator` instantiates an `oracle` state variable, which is the only address that is allowed to report payouts for questions. The oracle is intended to be the `UmaCtfAdapter`, which on its own is capable of resolving questions via integration with UMA's Optimistic Oracle and interacting directly with the Conditional Token Framework. Due to the `UmaCtfAdapter` already being in use as a standalone contract, its integration with the `NegRiskOperator` was unclear. Additionally, the following expected control flows were not documented: creating a market, proposing a question, resolving a question, and finalizing the payouts when the oracle for the `NegRiskOperator` is the `UmaCtfAdapter` and the Conditional Token Framework for the `UmaCtfAdapter` is the `NegRiskOperator`.

Consider documenting the following scenarios with a brief description of the expected behavior, who the intended caller is, and when the corresponding function is expected to be called:

- Preparing a market
- Preparing a question
- Resolving a question
- Reporting a payout for a resolved question
- Finalizing the payout of a resolved question

Consider adding documentation to both the `uma-ctf-adapter` and `neg-risk-ctf-adapter` codebases. Additionally, consider adding it to user-facing documentation.

***Update**: Partially resolved in pull request #72 at commit 392fa7e. Documentation for all five scenarios was not added.*

Low Severity

L-01 Admin Can Emergency-Resolve With Invalid Payouts Array

When a question is resolved, the `payouts` array reported to the CTF must be either `[1, 0]`, `[0, 1]`, or `[1, 1]`. However, during emergency resolution, this is never enforced on the admin-supplied `payouts` array.

Consider sanity-checking the `payouts` array to ensure that correct values are reported to the CTF.

***Update**: Resolved in pull request #63 at commit bc6938f.*

L-02 `getExpectedPayouts` Can Return Erroneous Values

`getExpectedPayouts` attempts to compute the expected payout of a question by retrieving the resulting price from the Optimistic Oracle, and then constructing the result into the defined payout array. However, `getExpectedPayouts` can return erroneous values for questions under the following circumstances:

- The question has not yet been resolved on the `UmaCtfAdapter`, but has been resolved in the Optimistic Oracle. In this circumstance, `getExpectedPayouts` would return the payouts array prematurely.
- The question has been flagged or paused on the `UmaCtfAdapter`, but has been resolved in the Optimistic Oracle. In this circumstance, `getExpectedPayouts` would return a payouts array for a question that should not be able to be resolved in the `UmaCtfAdapter` without admin intervention.
- The question has gone through an emergency resolution in the `UmaCtfAdapter`, but the question has been resolved separately in the Optimistic Oracle. In this circumstance, `getExpectedPayouts` may return a value that does not coincide with the payouts set by the admin.

Consider reverting the `getExpectedPayouts` call if the passed-in `questionId` corresponds to a question that has not yet been resolved, has been flagged or paused, and or has been emergency-resolved.

***Update**: Partially resolved in pull request #64 at commit 00918e7. The Polymarket team stated:*

> *Did not fix the case in which the question is yet to be resolved and `getExpectedPayouts` returns the payout array. This is the expected behavior.*

L-03 User Can Force a Dispute On An Emergency-Resolved Question

`emergencyResolve` enables the admin of the `UmaCtfAdapter` to record its own payouts for a given question. This in turn resolves the question, regardless of the status of the

corresponding price request on the Optimistic Oracle. However, the `priceDisputed` callback, which is called by the Optimistic Oracle in the event that a user disputes a price request proposal, does not check whether a question has already been resolved.

Therefore, it is possible for a user to dispute a price request on the Optimistic Oracle that corresponds to a question on the `UmaCtfAdapter` that has been emergency-resolved. Consequently, this may cause a new price request to be made on the Optimistic Oracle for the question, and change its timestamp. Ultimately this does not enable the price to be changed on the Conditional Token Framework, as the price can only be recorded once. However, this may cause confusion for users as events would be emitted, such as the `QuestionReset` event, that appear to indicate that an already-resolved question is being reset.

Consider adding a check to the `priceDisputed` function to prevent price request resets on questions that have already been resolved. In this case, `priceDisputed` should return the reward tokens back to the creator of the question and then return.

***Update**: Resolved in pull request #64 at commit 19fb877.*

Notes & Additional Information

N-01 Constants With Undocumented Literal Values

Throughout the codebase, there are several occurrences of constants which are assigned undocumented literal values. Considering documenting the origin of `YES_OR_NO_QUERY` and the meaning behind `8139`.

***Update**: Resolved in pull request #66 at commit 646e574.*

N-02 Constants Not Using `UPPER_CASE` Format

Throughout the codebase there are constants not using `UPPER_CASE` format. For instance:

- The `emergencySafetyPeriod` constant declared on line 38 in `UmaCtfAdapter.sol`

- The `yesOrNoIdentifier` constant declared on line 41 in `UmaCtfAdapter.sol`
- The `maxAncillaryData` constant declared on line 44 in `UmaCtfAdapter.sol`
- The `initializerPrefix` constant declared on line 5 in `AncillaryDataLib.sol`

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

***Update**: Resolved in pull request #67 at commit 0f6e7cb.*

N-03 The Admin Can Interact With Resolved Questions

The admin of the `UmaCtfAdapter` can interact with the state of resolved questions in the following ways:

- The admin can flag questions that have already been resolved.
- The admin can pause questions that have already been resolved.

In both cases, flagging and pausing a question that has already been resolved does not enable the admin to change the price of the question. However, it does allow admin-only events to be emitted, which could confuse users.

Consider adding a check to both the `flag` and `pause` functions in `UmaCtfAdapter` that reverts if the passed-in question has already been resolved.

***Update**: Resolved in pull request #68 at commit d4f1a27.*

N-04 The Admin Cannot Unflag Questions

`flag` enables the admin of the `UmaCtfAdapter` to mark questions for emergency resolution. This is done by setting the `emergencyResolutionTimestamp` for the passed-in `questionId`. However, the admin is unable to reset the `emergencyResolutionTimestamp` back to zero. Thus the admin is unable to unflag a question for emergency resolution.

Consider adding an administrative function that enables an admin to reset the `emergencyResolutionTimestamp` to zero.

***Update**: Resolved in pull request #69 at commit dc4c932.*

N-05 Unnecessary Dependency

The `ReentrancyGuard` dependency is unnecessary as it is only imported and inherited in the `UmaCtfAdapter` but never used. Consider removing the `ReentrancyGuard` contract as a dependency.

***Update**: Resolved in pull request #70 at commit fa63de9.*

Conclusions

One high-severity issue was found, and several other changes were proposed to follow best practices and reduce the potential attack surface. Additionally, suggestions have been made to increase the clarity of the interactions between the `UmaCtfAdapter` and the `NegRiskOperator`.