OpenZeppelin | security

Polymarket Multi-Outcome Markets 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	20 (15 resolved, 1 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	6 (5 resolved)
Notes & Additional Information	10 (6 resolved, 1 partially resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the `Polymarket/neg-risk-ctf-adapter` repository at the fda8085 commit.

In scope were the following contracts:

```
src
├── NegRiskAdapter.sol
├── NegRiskOperator.sol
├── Vault.sol
├── WrappedCollateral.sol
├── libraries
│   ├── Helpers.sol
│   └── NegRiskIdLib.sol
├── modules
│   ├── Admin.sol
│   ├── Auth.sol
│   └── MarketDataManager.sol
└── types
    └── MarketData.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. While the currently deployed Polymarket system exists only for binary markets, the contracts that were audited are designed to unify mutually exclusive binary markets into a single multi-outcome market structure. The resulting market includes a set of YES/NO questions of which one and only one will resolve as true. The two possible outcomes are tokenized as YES and NO tokens and backed by collateral, using Gnosis's Conditional Tokens Framework. Based on the outcome of the question, the YES and NO tokens will be redeemable for 0 or 1 unit of collateral. Below is an overview of some of the main contracts in the system.

`NegRiskAdapter` Contract

The `NegRiskAdapter` contract is designed as a wrapper around Gnosis's Conditional Tokens Framework. It allows anyone to create a multi-outcome market, specifying some metadata about the market and a fee in basis points, to be taken when positions are converted. Only the creator of a market can prepare questions for it and report outcomes.

Note that `prepareMarket` is a permissionless function, so a malicious user could create a market and some corresponding questions, and after victims have made enough bets, the attacker could report any outcome they desire. However, Polymarket intends to only represent and endorse markets created by the internal team, where the outcome is reported by an oracle, namely the UMA CTF Adapter system.

Given a question, a caller can split collateral into both YES and NO tokens and also merge YES and NO tokens to get collateral back. One collateral will be split into one YES and one NO tokens, and the two tokens can be merged to get one collateral back.

The fact that only one question can evaluate to YES guarantees equivalences among certain sets of positions. Therefore, `NegRiskAdapter` introduces the `convertPositions` function, which allows the caller to convert NO tokens corresponding to a set of questions to YES tokens corresponding to the complementary set of questions.

For example, consider a multi-outcome market representing a political election with candidates A, B and C (in the smart contracts, this would be one market with 3 questions), and a user having a position of 1 NO A and 1 NO B. The possible outcomes are:

- A wins, and the position is worth 1 collateral because of the NO B token.
- B wins, and the position is worth 1 collateral because of the NO A token.
- C wins, and the position is worth 2 collateral because of the NO A and NO B tokens.

Hence, an equivalent position to 1 NO A and 1 NO B would be 1 collateral and 1 YES C token. The `convertPositions` function is designed precisely for this, to convert a position into an equivalent one and refund collateral which is guaranteed to be won upon resolution.

For each position that is converted, Polymarket receives `feeInBps` basis points of released collateral and converted YES tokens. The fees are sent to the `Vault` contract, from where they can be withdrawn by one of the Vault's admins.

After the outcomes are reported and the payouts are registered in the CTF contract, a user can call `redeemPositions`, redeeming YES and NO tokens for collateral. Note that if redeeming both YES and NO tokens for the same question, only the tokens corresponding to the correct resolution are burned in the CTF, while the losing tokens remain in `NegRiskAdapter`.

`NegRiskOperator` Contract

The `NegRiskOperator` contract is intended as a permissioned operator for the `NegRiskAdapter` contract. An admin can set the oracle responsible for reporting the outcomes of questions, and prepare markets and questions.

The oracle is the only address allowed to report outcomes for questions, and once an outcome is reported, anyone can call `resolveQuestion` in order to propagate the result to the `NegRiskAdapter`.

As a safety mechanism, an admin can flag and unflag questions. Flagging is intended to block a question from being resolved, if there are suspicions that the oracle will report an incorrect result. A flagged question can be emergency-resolved by an admin, and the admin would provide what they consider to be the correct result for the question.

`WrappedCollateral` Contract

The `WrappedCollateral` contract represents an ERC-20 token that holds an underlying token (USDC) that is deposited by users when creating positions in the market. Namely, when

positions are split in `NegRiskAdapter`, collateral will be transferred from the user's balance and an identical amount of `WrappedCollateral` (WCOL), YES and NO tokens will be minted. Note that WCOL is never transferred to the user and is only ever held by the `NegRiskAdapter`, which creates and owns the `WrappedCollateral` contract. Collateral can only be released through the processes of merging, converting, or redeeming CTF token positions. There is no exchange rate explicitly computed between WCOL and USDC and one WCOL is always redeemable for 1 USDC. Despite this, the supply of WCOL may not match the balance of USDC held in the contract. During `convertPositions`, additional collateral is minted to cover the additional YES tokens minted, with no corresponding deposit. This unbacked WCOL remains in the `NegRiskAdapter`, seemingly locked forever. However, if there ever turns out to be a way to retrieve this extra WCOL balance, it can be unwrapped in the `WrappedCollateral` contract to steal underlying tokens from depositors.

Security Model and Trust Assumptions

- The ERC-20 used as collateral in `NegRiskAdapter` is expected to be set to USDC.
- The `CTHelpers` library is part of the Conditional Tokens Framework and is out of scope for this audit. It is expected to work correctly.
- Admins are expected to not be malicious. They will prepare markets with a reasonable `feeInBps`, as well as promptly flag questions with potentially incorrect 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 Negative Risk CTF Adapter system is not designed to support ties, and hence it is expected to only be used for questions that will never tie.
- 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. However, this means that there is no way to rescue users' investments in case a market becomes stuck and unresolvable. If a market cannot be resolved, then the YES/NO tokens are never redeemable.
- Even though just one question can resolve to true, each question must be resolved by setting the correct payouts for YES/NO. There is no guarantee that payouts will be set for all the questions, therefore it is implicitly assumed that all questions in a market will be resolved.

Privileged Roles

`NegRiskAdapter`:

- Anyone can create a market, but only the creator of a market can prepare questions for it and report outcomes.
- Anyone can split, merge or redeem positions for any condition in the linked CTF contract.
- Anyone can convert positions corresponding to a market that has been prepared.

`NegRiskOperator`:

- Admins can:
  - Set the immutable oracle.
  - Prepare a market and its questions.
  - Flag/unflag a question, in order to temporarily block question resolution.
  - Emergency-resolve a flagged question.
- Only the oracle can report payouts for a question.
- Anyone can resolve a question, as long as the payouts are available and the question is not flagged.

`Vault`:

- Admins can transfer any ERC-20 or ERC-1155 tokens owned by the vault.

`WrappedCollateral`:

- The owner can wrap collateral into `WrappedCollateral`. They can also burn and mint `WrappedCollateral`, or release collateral to an arbitrary address.
- Anyone can unwrap `WrappedCollateral` for collateral.

Critical Severity

C-01 Collateral Inside `WrappedCollateral` Contract Can Be Fully Drained

The `convertPositions` functionality aims to convert a set of NO tokens to a complementary set of YES tokens, while also potentially releasing some collateral.

For example, considering a market with 3 questions (Q1, Q2, Q3). A user can convert 100 NO Q1 and 100 NO Q2 tokens into 100 YES Q3 tokens and 100 USDC. Because only one question can evaluate to YES, the two positions are equivalent: answering NO to the first and second questions is the same as answering YES to the third question. 100 USDC of collateral is released since it is guaranteed that at least Q1 or Q2 will evaluate to NO, if not both.

When converting positions, the user's NO tokens are burned, which in the above example would be 100 NO Q1 and 100 NO Q2. In order to give YES tokens back, the protocol mints additional `WrappedCollateral` and splits it into YES and NO tokens. In the above example, `NegRiskAdapter` would receive 100 Q3 YES and 100 Q3 NO tokens. The YES tokens are correctly sent to the user, while the NO tokens are left inside the `NegRiskAdapter`.

This allows the user to later redeem the NO tokens as well, by calling `redeemPositions` with any value for `_amounts`, because the CTF contract redeems the caller's entire balance, and NegRiskAdapter unwraps all the accumulated `WrappedCollateral`. This makes the conversion of positions no longer equivalent: after the outcomes are reported, a user can redeem the NO tokens left in `NegRiskAdapter`.

Note that this can be exploited regardless of the market being set up by the `NegRiskOperator`. Any user can set up their market and questions, report the outcomes, and by repeating the above process, fully drain the collateral underpinning all markets, making them insolvent.

When converting positions, consider burning the NO tokens that result from splitting the newly minted `WrappedCollateral`.

***Update**: Resolved in pull request #6 at commit e49f076. NO tokens held by the `NegRiskAdapter` are now sent to the `noTokenBurnAddress` when converting a position from NO to YES.*

High Severity

H-01 Condition Prepared in CTF Prevents Further Questions From Being Added to a Market

When adding a question in a market, the condition is prepared in the CTF using a `questionId` composed of the `marketId` and the index of the new question. This makes it possible to front-run `NegRiskAdapter.prepareQuestion` and prepare the condition directly in the CTF using the same `questionId`. When this happens, the question can no longer be prepared in `NegRiskAdapter` as the `questionId` is already used, which prevents the number of questions in the market from incrementing. Hence, an attacker can prepare the condition directly on the CTF using the next `questionId` to prevent any further questions from being added to a market.

Consider preparing a market and all-encompassing questions in a single, atomic transaction.

***Update**: Resolved in pull request #7 at commit 6ed7b38.*

Medium Severity

M-01 Questions Can Be Resolved With Incorrect Payouts

After the payouts for a question are reported, `delayPeriod` seconds must pass until anyone can propagate the results to the Conditional Tokens Framework (CTF) by calling `resolveQuestion`.

`delayPeriod` is intended as a safety mechanism; in case the oracle reports incorrect payouts, admins have some time to notice and flag the question, effectively blocking propagation to the CTF. They can then investigate and report the correct payouts by calling `emergencyResolveQuestion`.

Given that the current `delayPeriod` is set to only 2 hours, there is a considerable possibility that a dishonest user can propagate incorrect payouts to the CTF before an admin is notified and is able to flag the question.

Consider increasing the `delayPeriod` to give admins more time to intervene.

***Update**: Resolved in pull request #8 at commit 78d8eae.*

M-02 `convertPositions` Assumptions Fail if the Outcome for a Market Is a Tie

`convertPositions` enables users to convert their NO tokens for one or more questions into a complementary set of YES tokens for the remaining questions in a market. Converting NO tokens to YES tokens for questions in a market is only equivalent so long as only one question resolves to 'Yes' and the remaining questions resolve to 'No.' However, it is possible for all questions in a market to resolve to 'No,' which would indicate a tie has occurred or that the result of the market was unable to be determined. In the case of a tie, the operating assumptions in `convertPositions` would no longer hold true. This is because a market where none of the questions resolve to 'Yes' can never have a complementary set of YES tokens for any combination of NO tokens.

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, as well as 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**: Resolved in pull request #9 at commit 70d6fa9. An explanation was added in the README that clearly states that a market must never result in either a tie or undetermined state, and that one question in the market must resolve to `yes`/`true`.*

Low Severity

L-01 Events Can Be Emitted Unnecessarily

The `addAdmin` function does not check if the address is not already an admin, and emits the `NewAdmin` event nonetheless. The `removeAdmin` function does not check if the address is an admin, and emits the `RemovedAdmin` event nonetheless.

This can make off-chain monitoring more complex and error-prone. Consider assigning to the `admins` mapping and only emitting events when necessary.

***Update**: Acknowledged, not resolved. The Polymarket team stated:*

> *Risk Accepted.*

L-02 `PositionsConverted` Event Is Emitted for Invalid Conversion

In the `NegRiskAdapter` logic, converting 0 amount of NO tokens for the complementary YES tokens is not a valid use case. However, when a user calls `convertPositions` with `_amount` 0, an event is emitted even if no conversion happens.

Consider reverting instead to prevent this event from being picked up by off-chain components.

***Update**: Resolved in pull request #10 at commit 42ffa52.*

L-03 Invalid Payouts Array Can Be Reported in `NegRiskOperator`

When payouts are reported in `NegRiskOperator`, the check for an invalid payout array will pass as long as one of the members is zero and the other non-zero. This disagrees with the comment, which states that arrays other than `[1, 0]` or `[0, 1]` are invalid. In case the array has a 0 and another number greater than 1, `result` will always evaluate to false. Consider reverting for all cases of invalid `_payouts`.

***Update**: Resolved in pull request #11 at commit 2f1a788.*

L-04 The `Admin` Contract is Unused

The intent of the `Admin` contract is to be able to pause/unpause individual questions as well as globally pause/unpause all questions. However, the contract is not inherited by any other contract.

Consider removing the `Admin` contract as it is unused.

***Update**: Resolved in pull request #12 at commit b1ed9b7.*

L-05 Empty `fallback` Function Allows Any Function Call To Succeed

The `NegRiskOperator` contract contains an empty `fallback` function, which is used to enable integration with oracles that expect to be able to call functions such as `prepareCondition`. The functionality required by these external oracles is handled in separate functions within the `NegRiskOperator`, and as such, the empty `fallback` function enables the calls to return without failing. However, allowing any function call to succeed against the `NegRiskOperator` is an anti-pattern, and could mask incorrect behavior.

Consider explicitly specifying which functions are allowed to 'No-Op' in the `NegRiskOperator` contract, and remove the empty `fallback` function.

***Update**: Resolved in pull request #13 at commit ba3d702. This was resolved by removing the `fallback` function and adding a `prepareCondition` function.*

L-06 Missing Custom Error in `revert` Statement

Within the `determine` function in `MarketData.sol`, there is a `revert` statement on line 55 that lacks a custom error.

Consider reverting with a custom error to be consistent with the rest of the codebase and avoid potential confusion when the contract reverts.

***Update**: Resolved in pull request #14 at commit 7e0261f.*

Notes & Additional Information

N-01 Inconsistent Function Naming

Throughout the codebase, a leading underscore is used to denote `private` and `internal` functions. However, the functions in `Helpers.sol`, `NegRiskIdLib.sol` and `MarketData.sol` do not follow this pattern.

To avoid confusion, consider enforcing the same function naming conventions across the entire codebase.

***Update**: Acknowledged, not resolved. The Polymarket team stated:*

> *We prefer internal function names in libraries to not be preceded by underscores.*

N-02 Inconsistent Function Ordering in Contracts

Throughout the codebase, there are multiple contracts that deviate from the Solidity Style Guide and have inconsistent ordering:

- The `CTHelpers` library.
- The `MarketStateManager` contract.
- The `NegRiskAdapter` contract.

To improve the project's overall legibility, consider standardizing ordering throughout the codebase, as recommended by the Solidity Style Guide - Order of Functions.

***Update**: Acknowledged, not resolved. The Polymarket team stated:*

> *We prefer the current function ordering for legibility.*

N-03 Inconsistent Pragma Statements

The files in the codebase use 3 different pragma statements: `0.8.19`, `^0.8.15`, and `>=0.5.1`.

It is always recommended to use only one, fixed pragma statement. This prevents accidentally compiling and deploying contracts with an unintended Solidity version, which can introduce unexpected bugs.

Consider reviewing and updating the pragma statements of all contracts to the same compiler version used in the testing phase.

***Update**: Acknowledged, not resolved. The Polymarket team stated:*

> *Top-level functions all have pragma 0.8.19.*

N-04 Typographical Errors

Consider correcting these typos to improve the readability of the codebase:

- paused should be "pause".
- 1_00_00 should be "10_000". This happens in several places across the codebase.

***Update**: Resolved in pull request #15 at commit 4b2459d.*

N-05 Unused Variables

In the `NegRiskAdapter` contract, the `feeDenominator` and `reportedAt_` variables are unused.

Consider removing them to improve the clarity of the codebase.

***Update**: Resolved in pull request #16 at commit 0ef736d.*

N-06 Gas Inefficiencies

There are several places across the codebase where changes can be made to improve gas consumption. For example:

- Refunding collateral in `convertPositions` can reuse the `payout` variable.
- In the `reportPayouts` function, `result` can be reused.

***Update**: Resolved in pull request #17 at commit 751af4f.*

N-07 Potential Unsafe ERC-20 Transfers

The ERC-20 standard allows for tokens to either throw an error or return `false` if a transfer fails. Transfers occur in the following places:

- Line 123 in `NegRiskAdapter.sol`
- Line 26 in `Vault.sol`
- Line 55 in `WrappedCollateral.sol`
- Line 67 in `WrappedCollateral.sol`
- Line 90 in `WrappedCollateral.sol`

The token transfers occur on the `WrappedCollateral` contract and on USDC (as the underlying ERC-20 token). Both of these contracts will throw an error in the event of a transfer failure. The risk of an ERC-20 token returning `false` instead of throwing an error is present and can lead to exploitation if more underlying ERC-20 tokens are allowed to be used as `WrappedCollateral`. The ERC-20 standard states:

> *The ERC-20 standard specifically notes "Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!".*

Consider using the OpenZeppelin `SafeERC20` library's `safeTransfer` function to transfer ERC-20 tokens from one address to another.

***Update**: Resolved in pull request #18 at commit d412e5d.*

N-08 TODO Comments in the Code

The following instances of TODO comments were found in the codebase.

These comments should be tracked in the project's issue backlog and resolved before the system is deployed:

- The `TODO` comment on line 101 of `NegRiskAdapter.sol`

Consider removing all instances of TODO comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO to the corresponding issues backlog entry.

***Update**: Partially resolved in pull request #19 at commit 1e17890. The `TODO` comment was removed from the codebase, however the external view helpers mentioned in the comment were not added and an issue was not created in the GitHub repo to track this.*

N-09 Unused Imports

Throughout the codebase there are imports that are unused and could be removed. For instance:

- Import `Admin` of `NegRiskAdapter.sol`
- Import `NegRiskAdapter` of `Vault.sol`
- Import `IUmaCtfAdapter` of `Vault.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

***Update**: Resolved in pull request #20 at commit bca1376.*

N-10 Constants Not Using `UPPER_CASE` Format

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

- The `noTokenBurnAddress` constant declared on line 50 of `NegRiskAdapter.sol`
- The `feeDenominator` constant declared on line 51 of `NegRiskAdapter.sol`
- The `delayPeriod` constant declared on line 50 of `NegRiskOperator.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 #21 at commit 08fe1a7.*

Conclusions

A few high-severity issues were found during the audit that impacted user funds and the functioning of the protocol. Additionally, several changes were proposed to follow best practices and reduce the potential attack surface. The Polymarket team has been very responsive and helpful in solving uncertainties around the system.