PUBLIC

Code Assessment

of the NegRiskAdapter Smart Contracts

11 April, 2024

Produced for

Polymarket

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Polymarket with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of NegRiskAdapter according to Scope to support you in forming an opinion on their security risks.

Polymarket implements an adapter contract that plugs between a conditional token exchange and the actual conditional tokens contract to enable prediction markets with multiple binary questions where exactly one question resolves to YES while all other questions resolve to NO. Additionally, an auxiliary contract is implemented that permissions the question creation.

The most critical subjects covered in our audit are correct accounting and access control. All covered subjects provide a high level of security.

It is worth to mention that the ambiguous guidelines for creating questions can lead to problematic cases in certain circumstances as can be seen in Emergency resolution mechanism possibly not sufficient.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the NegRiskAdapter repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	13 February 2024	e206dd2ed5aa24cf1f86990b875c6b1577be25e2	Initial Version

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

The following files were in scope:

- src/NegRiskAdapter.sol
- src/NegRiskCtfExchange.sol
- src/NegRiskFeeModule.sol
- src/NegRiskOperator.sol
- src/Vault.sol
- src/WrappedCollateral.sol
- src/libraries/CTHelpers.sol
- src/libraries/Helpers.sol
- src/libraries/NegRiskIdLib.sol
- src/modules/Auth.sol
- src/modules/MarketDataManager.sol
- src/types/MarketData.sol

2.1.1 Excluded from scope

Any contracts inside the repository that are not mentioned in `Scope` are not part of this assessment. All external libraries and imports are assumed to behave correctly according to their high-level specification, without unexpected side effects.

Tests and deployment scripts are excluded from the scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Polymarket offers an adapter (and some peripheral contracts) for Gnosis Conditional Tokens (CTF) that allows to group multiple binary (YES or NO) prediction questions into a combined market in which exactly one of the questions can resolve to YES while all other questions resolve to NO. Furthermore, it allows for the conversion of any NO tokens to the corresponding YES tokens of all other questions in the market at any time.

Questions are resolved through UMA oracles that allow for disputes during a challenge period. Still, it is possible that all questions of a market might be resolved to NO or that multiple questions are resolved to YES by the oracle. For this reason, markets used by Polymarket are created with a special contract that only allows permissioned accounts to create markets. These markets are then subject to an emergency resolution mechanism that enables the permissioned accounts to resolve questions in case of such misaligned outcomes.

Due to limitations with the underlying Conditional Tokens Framework the adapter is built on a special dummy token, used as collateral to allow for the minting of additional outcome tokens.

The tokens of each question in a market can be traded on an order-book based marketplace on which an operator executes orders received off-chain.

The parts of the system are described in detail in the following:

2.2.1 ConditionalTokens

The contracts reviewed in this report build on Gnosis' `ConditionalTokens` contract. In short, the contract offers the ability to create prediction markets (conditions) with an arbitrary amount of outcomes and arbitrary assignment of payout shares to these outcomes by an oracle account. Conditions can be used together with arbitrary collateral tokens. Conditions can also be chained together. The functionality is achieved with the following building blocks:

- `prepareCondition()` allows an oracle to create a condition for a specific question with predefined outcomes.
- `splitPosition()` allows a user to add collateral to a condition, minting 1 token of each possible outcome per collateral token.
- `mergePositions()` allows a user to burn tokens of all possible outcomes of a condition to receive the same amount of collateral.
- `reportPayouts()` allows the oracle of a condition to report a payout vector once the associated question can be resolved.
- `redeemPositions()` allows a user holding one or more outcome tokens to redeem them for collateral (as long as they have a payout vector associated).

For a more in-depth analysis of the contract, please refer to our Conditional Tokens Smart Contract Audit Report.

2.2.2 NegRiskAdapter

The `NegRiskAdapter` is used as a proxy to the Conditional Tokens contract that is widely used in the Polymarket ecosystem. It works on the CTF using `WrappedCollateral` as the collateral token of all conditions. The functions `splitPosition()`, `mergePositions()` and `redeemPositions()` forward the calls to the same functions in the CTF contract and convert the collateral to/from ``WrappedCollateral `` along the way.

Markets consisting of multiple questions are created by an oracle using `prepareMarket()`. The oracle can then add multiple questions to the market using `prepareQuestion()` which also prepares a corresponding condition in the underlying CTF contract.

The function `convertPositions()` allows the holder of any NO tokens in a market to convert these tokens to YES tokens of all other available questions in the same market. If a user holds NO tokens for n (with $n > 1$) questions of a market, the tokens are additionally converted to $n - 1$ collateral tokens. This

is possible due to the fact that a NO token always has a full payout when any other question in the market resolves to YES (and hence 2 different NO tokens always have at least 1 full payout).

Once a question is resolved, the oracle can call `reportOutcome()` to determine if a question is either answered with YES or NO. Partial outcome payouts are not allowed. Additionally, the function does not allow more than one question in a market to be answered with YES.

2.2.3 WrappedCollateral

`WrappedCollateral` is an ERC-20 contract that acts as an adapter between the CTF contract and the actual collateral token. Because the `NegRiskAdapter` can convert certain NO positions to YES positions in other questions of a market, these YES positions have to be minted on the CTF contract which requires collateral. This contract can be used to perform these minting operations without locking up additional collateral. It allows the `owner` (the `NegRiskAdapter`) to `mint()` and `burn()` tokens, as well as `wrap()` underlying tokens into wrapped collateral tokens. Everyone can `unwrap()` these tokens to get back the underlying collateral because CTF tokens can also be redeemed directly on the CTF contract, transferring wrapped collateral to the user.

2.2.4 NegRiskOperator

Since not all possible markets can be formulated in a way that ensures only one of the associated questions will be answered with YES (or can be answered without a tie) and `NegRiskAdapter` allows anyone to create markets, the `NegRiskOperator` is a permissioned contract that allows certain `admins` to create markets that can be trusted. `prepareMarket()` and `prepareQuestion()` are used to call their counterparty in the `NegRiskAdapter` and save the question IDs in the contract.

A special `oracle` contract (`UmaCtfAdapter` that uses UMA in case there is a dispute with the resolution of a question) is allowed to call `reportPayouts()` which stores the outcome of a question in the contract. After a delay period of one hour anyone can then call `resolveQuestion()` to report the outcome to the `NegRiskAdapater`.

If a question is problematic (e.g., it is the second YES outcome in a market), the admins of the contract can call `flagQuestion()` in which case the payout cannot be reported to the `NegRiskAdapter` anymore. The admins can then decide to call `unflagQuestion()` and return to the normal workflow or call `emergencyResolveQuestion()` after another one hour delay to report a different outcome that solves the question according to the market rules.

2.2.5 NegRiskCtfExchange

Polymarket offers a `CtfExchange` for binary CTF tokens that is used by certain `operators` to facilitate the matching of orders that are stored off-chain. `NegRiskCtfExchange` extends this contract to be able to work with the `NegRiskAdapter` instead of the `ConditionalTokens` contract.

For more information about this contract, please refer to our Polymarket Exchange Smart Contract Audit Report.

It is worth noting that users must set full approval of their CTF tokens both to the exchange and to the `NegRiskAdapter` as the adapter's `safeTransferFrom()` additionally checks the approval to the caller before calling `safeTransferFrom()` on the CTF contract.

2.2.6 NegRiskFeeModule

Since the `CtfExchange` allows for taker orders to be converted to maker orders if not fully matched, and maker orders to be charged lower fees, a `FeeModule` is used as a proxy to the `matchOrders()` function of the exchange that refunds the difference between the fee levels back to the creator of such orders. `NegRiskFeeModule` extends this contract to be able to work with the `NegRiskAdapter` instead of the `ConditionalTokens` contract.

2.2.7 Vault

The `NegRiskAdapter` allows the oracle to set fees for a market that are charged in `convertPositions()`. These fees are sent to the `Vault` contract that gives its `admins` the ability to transfer out any ERC-20 or ERC-1155 tokens.

2.2.8 Roles & Trust Model

`NegRiskAdapter.safeTransferFrom()` is permissioned and can be called by the Polymarket multisig, the `CtfExchange` or the fee module. However, it still ensures that the `from` address has given permission to the multisig before commencing a transfer. Otherwise, the contract is completely unpermissioned.

The `NegRiskOperator` can only be used by permissioned accounts. These accounts are able to:

1. Create markets that might not have a correct outcome with respect to the market rules of the `negRiskAdapter` (this is true for any market created on the `NegRiskAdapter`).
2. Flag questions to stall question resolution.
3. Emergency resolve any question to a wrong outcome (except for multiple YES outcomes in a market which is not possible).
4. Change the `oracle` that resolves the questions.

We assume that the `NegRiskOperator` is always used in conjunction with the `UmaCtfAdapter` oracle contract.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Emergency Resolution Mechanism Possibly Not Sufficient Risk Accepted

5.1 Emergency Resolution Mechanism Possibly Not Sufficient

Design Low Version 1 Risk Accepted

CS-PMNRA-001

`NegRiskOperator` allows the admins of the contract to flag a question after it has been reported by the oracle. With the function `emergencyResolveQuestion()`, the admins are then able to set a different outcome for the respective question. This is required for several edge cases:

1. All questions of a market resolve to NO.
2. More than one question of a market resolve to YES.

Questions are, however, not resolved all at once. Some markets might contain questions that resolve one after another with prolonged time periods in between. In this case, the above cases might become evident only after some questions have already been resolved and their corresponding tokens been redeemed.

For example, a problem in a market in which all questions ultimately resolve to NO might only be discovered when the last question resolves to NO. In this case, holders of NO tokens in the already resolved questions have already redeemed their tokens. The admins now have two/three possibilities:

1. Let the last question resolve to NO which invalidates the position of users that previously converted their NO tokens to YES tokens using `convertPositions()`.
2. Wrongly resolve the question to YES which invalidates the position of users that hold NO tokens of the last question.
3. Add more questions to the market (e.g., possible in "event happens at X" kind of markets) which traps collateral in the CTF contract (see Question preparation at later point for details).

Risk accepted:

Polymarket accepts the risk with the following statement:

It is our opinion that for certain markets, the risk is acceptable that all questions resolve NO. It is true in that case that certain funds may be locked; however, users are fully aware of their net position before and after a convert operations, so these locked funds are not user funds.

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 No Safe ERC-20 Approval

Informational Version 1 Risk Accepted

CS-PMNRA-002

`NegRiskAdapter` calls the collateral token's `approve()` directly without a check of the return value. If the chosen collateral token does not revert in this function call, it is possible that the contract can be deployed without the correct approval being set up.

Risk accepted:

Polymarket accepts the risk with the following statement:

> In the unlikely event that some version of these contracts is planned to be deployed with such an ERC20 collateral token, it would be immediately evident, giving minimal testing, that the approvals were not set properly.

6.2 Outcome Reporting Batch Function

Informational Version 1 Risk Accepted

CS-PMNRA-003

Since many markets in the `NegRiskAdapter` can contain questions that resolve at the same time, a batch function for reporting outcomes might make sense.

Risk accepted:

Polymarket accepts the risk with the following statement:

> Batch reporting is a challenge due to the nature of the oracle requests. In the future, we would be interested in integrating an oracle solution which can accommodate the multi-outcome nature of neg risk markets.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Market ID Equal to First Question ID

Note Version 1

`NegRiskAdapter` creates market IDS by hashing the oracle address, the fee in BPS and some metadata and then setting the rightmost 8 bits to 0. Such IDs are then used to create question IDs by adding the index of the question to the reserved 8 bits. Since the first question has index 0, its ID is equal to the market ID.

7.2 Question Preparation at Later Point

Note Version 1

`NegRiskAdapter` (and `NegRiskOperator`) allow to create a market and then successively add questions to the market. This means that further questions can also be added to the market at a later point in time. This is problematic if `convertPositions()` for this market has already been used successfully before the new question is added. Consider the following example:

1. A market consists of three questions A, B and C.
2. For each question, 100 tokens of each outcome have been minted. A total of 300 collateral tokens have been deposited into the CTF contract.
3. A user owns 100 B-NO and 100 C-NO tokens. They decide to convert these tokens using `convertPositions()` which burns their NO tokens, mints 100 A-YES tokens and transfers 100 collateral tokens to the user.
4. An admin adds a new question D to the market for which 100 collateral tokens are split into 100 D-YES and 100 D-NO tokens.
5. D resolves to YES.
6. 100 D-YES tokens, as well as 100 A-NO tokens, can be redeemed for 200 collateral. No other tokens are eligible and yet, the contract holds 300 tokens in collateral.

This shows that questions that have been added after a `convertPositions()` call can trap tokens inside the CTF contract if they resolve to YES.

7.3 Short Reporting Delay

Note Version 1

`NegRiskOperator` defines a `DELAY_PERIOD` that gives the admins of the contract time to interfere after the oracle has submitted invalid outcomes. The period is currently set to 1 hour. It is therefore imperative that Polymarket has set up a robust reporting and emergency reaction infrastructure to be able to always act on problems in this short timeframe.