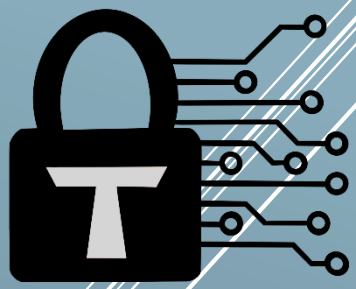
Trust Security



Smart Contract Audit

Butter Flat CFM

14/01/2025

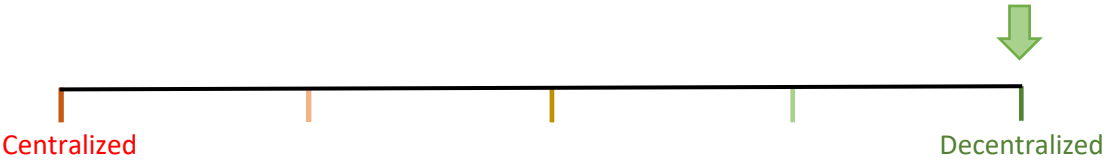
Executive summary

Category	Predication Markets
Audited file count	7
Lines of Code	521
Auditor	Trust
Time period	07/01/25-14/01/25

Findings

Severity	Total	Fixed	Acknowledged
High	1	-	-
Medium	1	-	-
Low	2	-	-

Centralization score



Signature

Digitally signed by Trust
Date: 2025.01.14 11:45:54 +02'00'

Document properties

Versioning

Version	Date	Description
0.1	14/01/25	Client report

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- FlatCFMFactory.sol
- ConditionalScalarMarket.sol
- FlatCFMRealityAdapter.sol
- Types.sol
- FlatCFM.sol
- FlatCFMOracleAdapter.sol
- String31.sol

Scope Exclusion List:

- Issues that have been reported in prior audit reports.
- Issues that already have attached PRs

Repository details

- **Repository URL:** https://github.com/butterygg/cfm-v1.git
- **Commit hash:** 4f1186ded77209bc42bfd98fc5d598994e5a311c

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys assessing bounty contests in C4 as a Supreme Court judge.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, reducing attack risks
Documentation	**Good**	Project is mostly very well documented.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Excellent**	Project does not introduce centralization risks.

Findings

High severity findings

TRST-H-1 Anyone can completely manipulate the results of the funding market

- **Category:** Access-control issues
- **Source:** FlatCFMRealityAdapter.sol
- **Status:** Open

Description

When there is an answer for the posted Reality questions, users are expected to call *resolve()* on the ConditionalScalarMarket or the FlatCFM. These call the reporting functions of the Adapter, which report the payout to the conditional tokens. The issue is that the *report()* calls pass data to the Adapter, which is interpreted as sensitive and trusted input. However, anyone may perform the *report()* calls directly on the Adapter, and feed malicious values. For *reportMetricPayouts(),* a user could send any **minValue** and **maxValue**, which can easily cause the wrong long/short token to gain value. For *reportDecisionPayouts()*, the input **outcomeCount** cannot be abused (since it has to equal the true outcome count), but as a best practice that function should also be protected.

Recommended mitigation

There are two approaches:

- Introduce access-control for the reporting functions. Only a market which is in the allowlist can call them.
- Fetch the sensitive data via pull from the factory, by looking up a given question ID. Some refactoring would be necessary to get it to work.

Team response

TBD

Medium severity findings

TRST-M-1 The RealityAdapter does not support arbitrator question fees and answer bounties

- **Category:** Logical Flaws
- **Source:** FlatCFMRealityAdapter.sol
- **Status:** Open

Description

In the RealityAdapter, the platform launches a question on RealityETH. On the other side, Reality expects value transfer which comprises of a **bounty** and an **arbitrator_question_fee**. However, since the value passed is always zero, there are two consequences:

- The platform can only operate with an arbitrator which does not charge any question fee. This limits current functionality and future interoperability.
- The creator cannot set a non-zero **bounty** to answer the question (which serves as an incentive for getting the question fulfilled).

Recommended mitigation

Allow the creator of a new market to pass in a positive value.

Team response

TBD

Low severity findings

TRST-L-1 Hard coded nonce of questions asked on Reality leads to unwanted effects

- **Category:** Logical flaws
- **Source:** FlatCFMRealityAdapter.sol
- **Status:** Open

Description

In Reality ETH, a question ID is built by hashing the contents as well as various properties of the question, including a **nonce** value. In the RealityAdapter, this value is hard-coded to be zero, both at the pre-calculation of questionID, and the parameter passed to *askQuestionWithMinBond()*. That is not necessarily an issue according to the documentation, but there are some consequences to hard-coding the nonce:

- Assuming M-1 is fixed and a bounty can be delivered to the question, an attacker may frontrun the question and specify a zero **bounty** value. The value passed to the question is not part of the resulting hash, so the *_askQuestion()* logic would determine the question has already been asked, and return the ID. The true intended bounty would not be consumed.
- If, though unlikely, a question with the same parameters would need to be asked (including same timestamp), it would not create a new question, although it may be intended to have a fresh one.

Recommended mitigation

Allow the user to pass an arbitrary nonce to the contract. Additionally, if the question already exists and a positive ETH value is sent to the contract, send the ETH to the bounty pot using *fundAnswerBountyERC20()*.

Team response

TBD

TRST-L-2 Use of unsafe ERC20 transfers in ConditionalTokens limits interoperability

- **Category:** ERC20 compliance issues
- **Source:** ConditionalTokens.sol
- **Status:** Open

Description

The CFM integrates with a self-deployed Gnosis ConditionalTokens contract. Unfortunately, the upstream code does not use the SafeERC20 library from OpenZeppelin, which prevents operability with several non-compliance classes of ERC20. This includes tokens like BNB, USDT and others which would fail to operate as collateral.

From discussions with the client, there is not yet a definite list of tokens which would be used as collateral, so it is highly recommended to not limit such use cases in the contract code.

Recommended mitigation

Consider using the SafeERC20 library.

Team response

TBD

Additional recommendations

TRST-R-1 Fix typos

The following names have typos:

- *deployWrappedConditiontalTokens()*

TRST-R-2 Improve documentation

- The *redeem()* function of the ScalarMarket lacks documentation of the invalid token.

TRST-R-3 Remove unused code

The **SEPARATOR** variable in RealityAdapter is never used, so it is best to remove it.

TRST-R-4 Coding style recommendations

- Consider capitalizing variable names for immutable and constant variables (including ones set via the *initialize()* function).
- Non-external functions should be prefixed with an underscore for clarity.

Centralization risks

The contracts don’t manage any privileged addresses and contain no centralization risks.

Systemic risks

TRST-SR-1 External integration risks

The contracts make use of several integration points, it should be clear that a compromise in the upstream code could affect the platform.

- Reality.ETH code and operations
- ConditionalTokens code
- ERC1155Factory code
- External tokens and AMMs that host CFM tokens

TRST-SR-2 Capital efficiency risks

A user that elects to participate in a particular long-short market would have to deposit collateral for the decision market. If there are X elected projects, a user participating in a single market would leave (X-1/X) of the collateral unused. This represents a risk of not being attractive enough to capture sufficient liquidity, and liquidity is critical for making the market trustable as a predictor of project success.