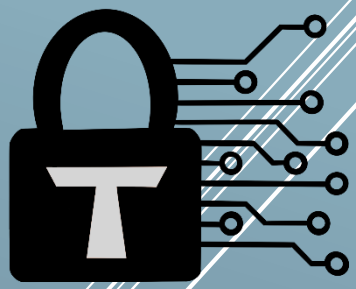
Trust Security

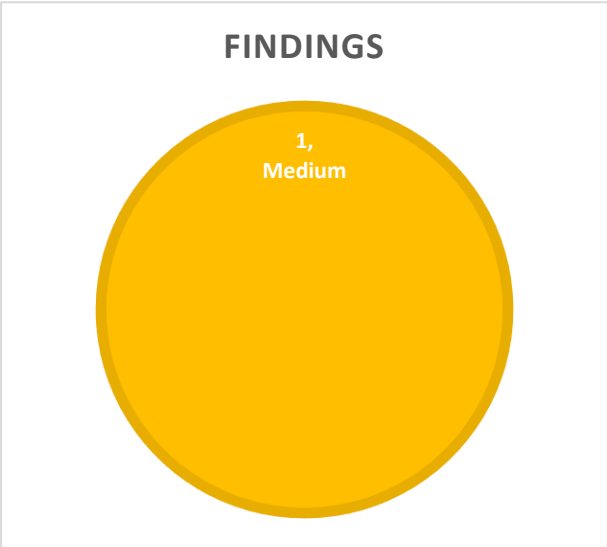


Smart Contract Audit

Butter – Invalidless Markets

29/04/25

Executive summary

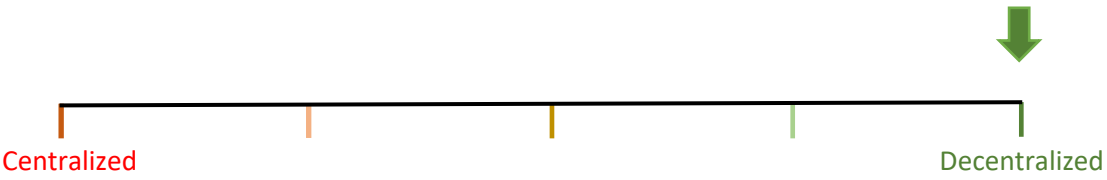


Category	Prediction Market
Audited file count	3
Lines of Code	238
Auditor	Trust
Time period	27/04/25-29/04/25

Findings

Severity	Total	Fixed	Acknowledged
High	0	-	-
Medium	1	-	-
Low	0	-	-

Centralization score



Signature

Digitally signed by Trust
Date: 2025.04.29 20:09:05 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	29/04/25	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

- invalidless/InvalidlessConditionalScalarMarket.sol
- invalidless/InvalidlessFlatCFMFactory.sol
- invalid/Types.sol

Repository details

- **Repository URL:** https://github.com/butterygg/cfm-v1
- **Commit hash:** 90da3eca4e2d7701e288a56e4cb3da47128fd1c0

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 sharing knowledge and experience with aspiring auditors through X or the Trust Security blog.

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	**Excellent**	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

Medium severity findings

TRST-M-1 An attacker could provide a malicious invalid payout distribution

- **Category:** Frontrunning attacks, access-control issues
- **Source:** InvalidlessFlatCFMFactory.sol
- **Status:** Open

Description

Butter integrates two types of markets. The scalar market, after the latest code change, supports payout in short or long tokens even on an invalid result. The market determines payout through the **defaultInvalidPayouts** parameter provided to *createConditionalScalarMarket()*. However, the function call is separate from the Flat decision market creation at *createFlatCFM()*. It is meant to support multiple calls to create a market for each outcome. Beforehand, there were no trust assumptions on the caller of scalar creation, because it received no parameters, instead it reads them from the Flat deployment. Now, the caller provides trusted input which will be pivotal in resolution of invalid payouts.

An attack could fill all new scalar markets with malicious outputs in order to profit from an invalid event. In case the honest deployer realizes the markets were already created, they would be forced to redeploy the market, and anyone holding tokens up to this point would be at risk.

Recommended mitigation

Pass the **defaultInvalidPayouts** in the *createFlatCFMFunction()* and store it in the storage mapping. It would then be safe to use in the scalar market creation logic.

Team response

TBD

Additional recommendations

TRST-R-1 Verify sane values for invalid payouts to avoid unresolvable outcomes

When passing **defaultInvalidPayouts**, it should be verified there is at least one non-zero value, and also that the sum of all payouts does not exceed **MAX_UINT256.** Otherwise, the *reportPayouts()* call when resolving the market would fail, which would brick the entire deposited collateral.

TRST-R-2 Remove unused code

The *_discreetPartition()* function in the invalidless condition market can be safely removed, as the functions which use it like *redeem()* and *merge()* have been removed.