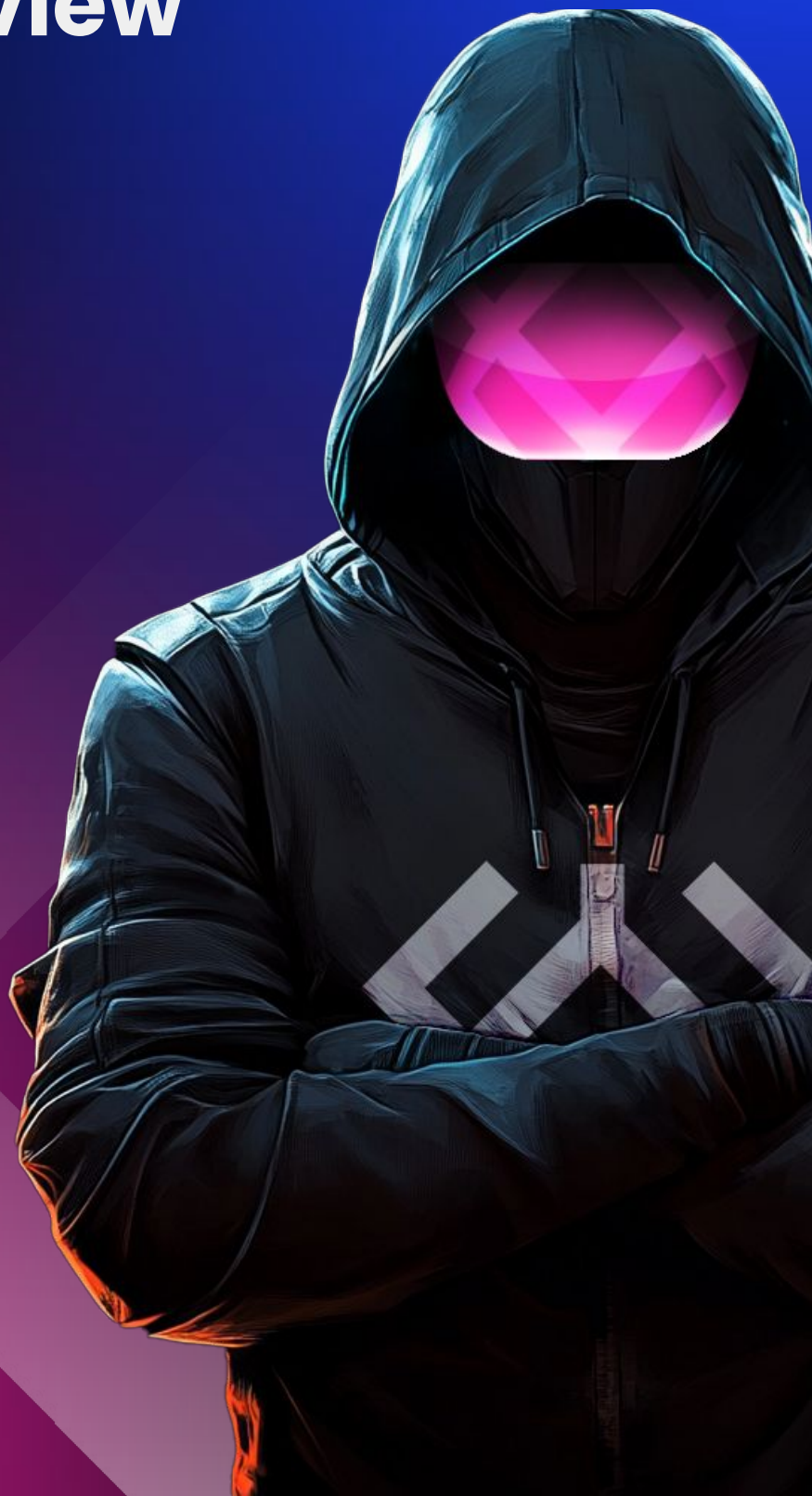
Butter

Mitigation Review

Report

February 2025



Report: 39495

FlatCFM cannot be resolved in case answer of questionId are in greater or equal to 2^OUTCOME_COUNT and answer % 2^OUTCOME_COUNT is 0

Report Date: 31/01/2025

Whitehat: perseverance

Program Action: Confirmed as low severity

Description: The ***resolve*** function is responsible for calculating payouts when the market is resolved. However, it didn't handle out-of-range answers properly, causing funds to be stuck in the contract. If the oracle returned a numeric answer greater than or equal to ***2^outcomeCount***, the payout calculation resulted in all zeroes, causing reportPayouts to revert due to ***require(den > 0, "payout is all zeroes");***

Mitigation Efforts:

Fix Description:

Commit link:
https://github.com/butterygg/cfm-v1/commit/178ee7f26b305ecb98be6684849a97c1a551c70d

An extra condition was added to handle out-of-range oracle answers. If ***numericAnswer*** is greater than or equal to ***2^outcomeCount***, it is marked as invalid by setting ***payouts[outcomeCount] = 1***. This prevents payout errors and ensures funds don't get locked.

JavaScript

```
if (numericAnswer == 0 || numericAnswer >= (1 << outcomeCount) ||
oracleAdapter.isInvalid(answer)) {
// 'Invalid' receives full payout
payouts[outcomeCount] = 1;
}
```

Immunefi

Review Methodology:

We used static analysis and unit tests to ensure that the bitwise check correctly handles cases where numericAnswer is greater than or equal to ***2^outcomeCount***

Review Findings:

Final Resolution Status: Resolved

The fix correctly flags out-of-range answers as invalid outcome, preventing the reportPayouts function from reverting and ensuring the market can resolve properly.

Unit Test:

```JavaScript
function testResolveOutOfRangeAnswer() public {
       uint256[] memory plainAnswer = new uint256[](OUTCOME_COUNT + 2);
        plainAnswer[0] = 0;
        plainAnswer[OUTCOME_COUNT - 1] = 0;
        plainAnswer[OUTCOME_COUNT] = 1;
        bytes32 answer = _toBitArray(plainAnswer);

        console.log("Bit Array:");
        for (uint256 i; i < plainAnswer.length; i++) {
             console.log("Index %s: %s", i, plainAnswer[i]);
          }
        console.log("OUTCOME_COUNT:", OUTCOME_COUNT);
        console.log("Numeric Answer:", uint256(answer));
        vm.mockCall(
            address(oracleAdapter),
            abi.encodeWithSelector(FlatCFMOracleAdapter.getAnswer.selector, QUESTION_ID),
            abi.encode(answer)
        );

        cfm.resolve();
    }
```

In the previous version, ***testResolveOutOfRangeAnswer()*** failed due to improper handling of out-of-range answers in the ***resolve()*** function

```
Ran 1 test for test/unit/FlatCFM.t.sol:TestResolve
[FAIL: revert: payout is all zeroes] testResolveOutOfRangeAnswer() (gas: 142183)
Logs:
  Bit Array:
  Index 0: 0
  Index 1: 0
  Index 2: 0
  Index 3: 0
  Index 4: 0
  Index 5: 0
  Index 6: 0
  Index 7: 0
  Index 8: 0
  Index 9: 0
  Index 10: 0
  Index 11: 0
  Index 12: 0
  Index 13: 0
  Index 14: 0
  Index 15: 0
  Index 16: 0
  Index 17: 0
  Index 18: 0
  Index 19: 0
  Index 20: 0
  Index 21: 0
  Index 22: 0
  Index 23: 0
  Index 24: 0
  Index 25: 0
  Index 26: 0
  Index 27: 0
  Index 28: 0
  Index 29: 0
  Index 30: 0
  Index 31: 0
  Index 32: 0
  Index 33: 0
  Index 34: 0
  Index 35: 0
  Index 36: 0
  Index 37: 0
  Index 38: 0
  Index 39: 0
  Index 40: 0
  Index 41: 0
  Index 42: 0
  Index 43: 0
  Index 44: 0
  Index 45: 0
  Index 46: 0
  Index 47: 0
  Index 48: 0
  Index 49: 0
  Index 50: 1
  Index 51: 0
  OUTCOME_COUNT: 50
  Numeric Answer: 1125899906842624

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 7.46ms (2.36ms CPU time)

Ran 1 test suite in 144.29ms (7.46ms CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/unit/FlatCFM.t.sol:TestResolve
[FAIL: revert: payout is all zeroes] testResolveOutOfRangeAnswer() (gas: 142183)

Encountered a total of 1 failing tests, 0 tests succeeded
```

Immunefi

After the fix, ***resolve()*** correctly processes out-of-range answers.

```
[PASS] testResolveOutOfRangeAnswer() (gas: 126804)
Logs:
  Bit Array:
  Index 0: 0
  Index 1: 0
  Index 2: 0
  Index 3: 0
  Index 4: 0
  Index 5: 0
  Index 6: 0
  Index 7: 0
  Index 8: 0
  Index 9: 0
  Index 10: 0
  Index 11: 0
  Index 12: 0
  Index 13: 0
  Index 14: 0
  Index 15: 0
  Index 16: 0
  Index 17: 0
  Index 18: 0
  Index 19: 0
  Index 20: 0
  Index 21: 0
  Index 22: 0
  Index 23: 0
  Index 24: 0
  Index 25: 0
  Index 26: 0
  Index 27: 0
  Index 28: 0
  Index 29: 0
  Index 30: 0
  Index 31: 0
  Index 32: 0
  Index 33: 0
  Index 34: 0
  Index 35: 0
  Index 36: 0
  Index 37: 0
  Index 38: 0
  Index 39: 0
  Index 40: 0
  Index 41: 0
  Index 42: 0
  Index 43: 0
  Index 44: 0
  Index 45: 0
  Index 46: 0
  Index 47: 0
  Index 48: 0
  Index 49: 0
  Index 50: 1
  Index 51: 0
  OUTCOME_COUNT: 50
  Numeric Answer: 1125899906842624

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.75ms (2.43ms CPU time)

Ran 1 test suite in 169.32ms (7.75ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total
tests)
```

Report: 39181

Bond Fund will be Lost When Question is Asked Again

Report Date: 24/01/2025

Whitehat: Topmark

Program Action: Confirmed as Insight (Security Best Practices)

Description: When a user submits a duplicate question, the ***FlatCFMRealityAdapter*** contract returns the existing ***questionID*** but does not reject the transaction. Since the function is payable, any msg.value sent with the request gets stuck in the contract with no way to refund it..

Mitigation Efforts:

Fix Description:

Commit link:
https://github.com/butterygg/cfm-v1/commit/0475c9f330f7f74a1a29d77d39496f52a5be7e5f

To fix this issue, the following check was added:

```javascript
if (oracle.getTimeout(questionId) != 0) {
if (msg.value > 0) revert QuestionAlreadyAsked();
return questionId;
}
```

Now, if the question has already been asked, the contract reverts the transaction if any funds were sent.

Review Methodology:

The fix was verified through static analysis and unit tests.

Review Findings:

Final Resolution Status: Resolved

We confirmed that the contract correctly refunds msg.value if the question already exists.

Immunefi

Unit Test:

JavaScript

```
function test_RefundsWhenMetricQuestionAlreadyAsked() public {
    address user = address(0x123);
    vm.deal(user, 3 ether);
    uint256 startingBalance = user.balance;
    console.log("Balance before first question:", user.balance);
    vm.startPrank(user);
    realityAdapter.askMetricQuestion{value: 1 ether}(
        metricTemplateId,
        genericScalarQuestionParams,
        "A"
    );
    vm.stopPrank();
    console.log("Balance after first question:", user.balance);
    vm.mockCall(
        address(reality),
        abi.encodeWithSelector(IRealityETHCore.getTimeout.selector),
        abi.encode(uint256(1))
    );
    vm.startPrank(user);
    console.log("Balance before duplicate call:", user.balance);
    vm.expectRevert(FlatCFMRealityAdapter.QuestionAlreadyAsked.selector);
    realityAdapter.askMetricQuestion{value: 1 ether}(
        metricTemplateId,
        genericScalarQuestionParams,
        "A"
    );
    vm.stopPrank();
    assertEq(user.balance, startingBalance - 1 ether);
    console.log("Final balance:", user.balance);
}
```

```
[PASS] RefundsWhenMetricQuestionAlreadyAsked() (gas: 105783)
Logs:
  Balance before first question: 3000000000000000000
  Balance after first question: 2000000000000000000
  Balance before duplicate call: 2000000000000000000
  Final balance: 2000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.83ms (1.86ms CPU time)

Ran 1 test suite in 143.06ms (6.83ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total
tests)
```

Report: 39271

Check `numericAnswer` before external call to check answer is valid or not

Report Date: 26/01/2025

Whitehat: iehnnkta

Program Action: Confirmed as Insight (Code Optimizations and Enhancements)

Description: The ***resolve()*** function determines whether the provided answer is invalid using two conditions. It first calls the external ***isInvalid()*** function to check if all bits in the answer are set to 1. When that check fails, it proceeds to evaluate numericAnswer. In cases where ***isInvalid()*** returns false, but numericAnswer is 0, the function makes an unnecessary external call, wasting gas on a more expensive operation before performing the simpler ***numericAnswer == 0 check***

```JavaScript
 if (oracleAdapter.isInvalid(answer) || numericAnswer == 0) {
// 'Invalid' receives full payout
payouts[outcomeCount] = 1;
}
```

Migration Efforts:

Fix Description

Commit link:
https://github.com/butterygg/cfm-v1/commit/178ee7f26b305ecb98be6684849a97c1a551c70d

The order of condition checks has been reversed. The external call to the isInvalid() function is now made only when numericAnswer is neither 0 nor greater than or equal to 2^outcomeCount

```JavaScript
 if (numericAnswer == 0 || numericAnswer >= (1 << outcomeCount) ||
oracleAdapter.isInvalid(answer)) {
// 'Invalid' receives full payout
payouts[outcomeCount] = 1;
}
```

Immunefi

Fix review

Final Resolution Status: Resolved

The code is now more gas-efficient by performing cheaper checks before making the external call.

Report: 39487

flatCfmImplementation and conditionalScalarMarketImplementation contracts can be initialized by anyone

Report Date: 24/01/2025

Whitehat: onthesunnyside

Program Action: Confirmed as Insight (Security Best Practices)

Description: When the FlatCFMFactory is deployed, it creates two uninitialized implementation contracts Because these contracts are permissionless, anyone can call their initialize function with arbitrary parameters.

Constructor in FlatCFMFactory.sol

```JavaScript
constructor(IConditionalTokens _conditionalTokens, IWrapped1155Factory _wrapped1155Factory) {

conditionalTokens = _conditionalTokens;

wrapped1155Factory = _wrapped1155Factory;

flatCfmImplementation = address(new FlatCFM());

conditionalScalarMarketImplementation = address(new ConditionalScalarMarket());

}
```

Migration Efforts:

Although the project accepted this bug as insight, no changes were made to the contract to fix the issue

Fix Review

Final Resolution Status: Not resolved / Acceptable Risk

When the FlatCFMFactory is deployed, it creates two uninitialized contracts FlatCFM and ConditionalScalarMarket, which act as templates. Although anyone can call their initialize functions, this does not affect the protocol or its clones, because neither the factory nor any cloned contract relies on the template's internal state. Each clone created by createFlatCFM has its own properly initialized parameters, so these uninitialized templates pose no material risk.
https://github.com/butterygg/cfm-v1/blob/main/src/FlatCFMFactory.sol#L121-L141

Report: 39524

Incorrect Outcome Formatting in Reality Adapter Leads to Wrong Number of Outcomes

Report Date: 24/01/2025

Whitehat: NHristov

Program Action: Confirmed as Insight (Documentation Improvements)

Description: The ***_formatDecisionQuestionParams*** function does not correctly handle outcome names with special characters like commas and quotes.
For example, an array like ***["Like Two "C"", "B"", "No"]*** gets converted to ***"Like Two "C", "B"","No"***, which Reality.eth interprets as ***"Like Two ", "C", "B"","No"***, leading to unintended extra outcomes.

Fix Review:

No code changes were made, as this validation will be handled on the frontend

Report: 39539

Insufficient validation of tokens when created in `PlayCollateralTokenFactory::createCollateralToken`

Report Date: 01/02/2025

Whitehat: Bx4

Description: The ***createCollateralToken*** function does not check whether the provided token ticker already exists before creating a new token

Migration Efforts:

The bug report was accepted as a best practice recommendation, but no code changes were made to resolve the issue.

Fix Review

Final Resolution Status: Not resolved / Acceptable Risk

This bug is considered an acceptable risk since the attack vector relies heavily on social engineering to trick users into interacting with a malicious token.

Immunefi

End of Report