Belief Market

Belief Market Audit Report

Review Date(s):
4/1/25 – 4/3/25
Fix Review Date(s):
4/7/25

Bio

As a professional smart contract auditor, I have conducted over 100 security reviews for public and private clients. With 30+ first-place finishes in public contests on platforms like Code4rena and Sherlock, I have been recognized as a top-performing security expert. By prioritizing rigorous analysis and providing actionable recommendations, I have contributed to securing over **$1 billion in TVL across 100+ protocols**. Throughout my career I have collaborated with many organizations including the prestigious Blackthorn as a founding security researcher and as a Lead Security researcher at SpearbitDAO.

Scope

The upredict-contracts repo was reviewed at commit hash 06da56c

In-Scope Contracts:

contract/*.sol

Deployment Chain(s):

Polygon Mainnet

Fix Review Commit Hash: 40f6609

Summary of Findings

Identifier	Title	Severity	Mitigated
[H-01]	Any bet can be refunded by supplying invalid betBlob	High	✔
[H-02]	Malicious request with amount == 0 can be used to drain market	High	✔
[M-01]	Changes to creatorFeeDecimal and operatorFeeDecimals will retroactively apply to older bets	Medium	✔

Detailed Findings

[H-01] Any bet can be refunded by supplying invalid betBlob

Details

MarketsBase.sol#L119-L146

```
    function requestRefund(BetRequest calldata request, BetBlob calldata
betBlob)
        external
        returns (IERC20 token, address to, uint256 amount)
    {
        RequestCommitment requestCommitment = getCommitment(request);
        BetState storage betState = bets[requestCommitment];
        require(betState.amount == request.amount,
MarketsBetDoesntExist(requestCommitment));
        betState.amount = 0;

        require(
            block.number >= request.refundStartBlock,
            MarketsRefundTooEarly(requestCommitment,
request.refundStartBlock, block.number)
        );

@>      MarketCommitment marketCommitment = _getMarketFromBet(betBlob);
        ResultCommitment resultCommitment = marketResults[marketCommitment];
        require(
            resultCommitment == nullResultCommitment,
MarketsResultAlreadyRevealed(marketCommitment, resultCommitment)
        );

        token = request.token;
        to = request.from;
        amount = request.amount;

        token.safeTransfer(to, amount);

        emit MarketsRefundIssued(requestCommitment, marketCommitment, token,
to, amount);
    }
```

When requesting a refund, the `marketCommitment` is retrieved from the `betBlob` data. However the commitment hash of the supplied `betBlob` is never validated against the `BetCommitment` stored in the `betRequest`. This allows any arbitrary `betBlob` to be supplied during the refund process. To exploit this, a malicious user can supply a `betBlob` that links to an

invalid market. This satisfies the `resultCommitment == nullResultCommitment` requirement, allowing users to receive an invalid refund and cause a shortfall in the protocol.

Lines of Code

MarketsBase.sol#L119-L146

Recommendation

The commitment hash of the supplied `betBlob` should be validated against the `BetCommitment` in the `betRequest`. This ensures that the `betBlob` corresponds to the originally committed `BetCommitment`, preventing the use of arbitrary or malicious `betBlob` data during refund requests.

Remediation

Fixed in commit 40f6609. The commitment of the supplied `betBlob` is now calculated and compared with the that of the `request` to ensure the supplied `betBlob` is legitimate.

[H-02] Malicious request with amount == 0 can be used to drain market

Details

MarketsBase.sol#L271-L282

```
    {
        BetState storage betState = bets[requestCommitment];
        BetCommitment betCommitment = getCommitment(betBlob);
        require(
            request.betCommitment == betCommitment,
            MarketsInvalidBetRequest(requestCommitment, betCommitment,
request.betCommitment)
        );
@>      require(betState.amount == request.amount,
MarketsBetDoesntExist(requestCommitment));


        // Since the bet is revealed, no amount should remain to be revealed
        betState.amount = 0;
    }
```

When revealing a bet, the above lines are designed to prevent invalid bets from being revealed. By checking that `betState.amount == request.amount` then setting `betState.amount` to `0`, it simultaneously prevents most invalid bets as well as double reveals. However, it misses the edge case in which `request.amount == 0`. This allows a malicious user to submit an invalid `betRequest` with `request.amount == 0` and a corresponding `betBlob` that will successfully bypass this check.

WeightedParimutuelMarkets.sol#L115-L139

```
    function _getPayout(
        MarketBlob calldata marketBlob,
        ResultBlob calldata resultBlob,
        BetRequest calldata request,
        BetBlob calldata betBlob
    )
        internal
        pure
        override
        returns (uint256 winningPotAmount, uint256 losingPotAmount, uint256
marketDeadlineBlock, address creator)
    {
        MarketInfo memory marketInfo = abi.decode(marketBlob.data,
(MarketInfo));
        BetHiddenInfo memory hiddenInfo = abi.decode(betBlob.data,
(BetHiddenInfo));
```

```
        ResultInfo memory resultInfo = abi.decode(resultBlob.data,
(ResultInfo));

        marketDeadlineBlock = marketInfo.deadlineBlock;

        creator = abi.decode(marketBlob.data, (MarketInfo)).creator;
        uint256 betOutcomeMask = (1 << hiddenInfo.outcome);
        if ((betOutcomeMask & resultInfo.winningOutcomeMask) != 0) {
            winningPotAmount = request.amount;
            losingPotAmount =
@>              Math.mulDiv(hiddenInfo.betWeight, resultInfo.losingTotalPot,
resultInfo.winningTotalWeight);
        }
    }
```

Since the `betBlob` is arbitrary, any value for `hiddenInfo.betWeight` can be supplied. As a result, the entire `resultInfo.losingTotalPot` can be stolen via this attack vector.

Lines of Code

MarketsBase.sol#L271-L282

Recommendation

`revealBet` should revert if `request.amount == 0`.

Remediation

Fixed in commit 40f6609. `revealBet` now requires that `request.amount > 0`

[M-01] Changes to creatorFeeDecimal and operatorFeeDecimals will retroactively apply to older bets

Details

MarketsBase.sol#L76-L80

```
function setFees(uint16 _creatorFeeDecimal, uint16 _operatorFeeDecimal)
external onlyRole(DEFAULT_ADMIN_ROLE) {
    creatorFeeDecimal = _creatorFeeDecimal;
    operatorFeeDecimal = _operatorFeeDecimal;
    emit MarketsFeesChanged(_creatorFeeDecimal, _operatorFeeDecimal);
}
```

MarketsBase.sol#L300-L314

```
    if (losingPotAmount > 0) {
        uint256 currentlyAvailable = availableLosingPot[marketCommitment];
        require(currentlyAvailable >= losingPotAmount,
MarketsInvalidResult(marketCommitment, resultCommitment));
        availableLosingPot[marketCommitment] = currentlyAvailable -
losingPotAmount;

        // only charge fees on the losing pot, to discourage markets that
        // are heavily imbalanced. If the losing pot is small (because it's
        // a very unlikely result), then creator fees are also small
@>      uint256 creatorFee = (creatorFeeDecimal * losingPotAmount) /
FEE_DIVISOR;
@>      uint256 operatorFee = (operatorFeeDecimal * losingPotAmount) /
FEE_DIVISOR;
        creatorFees[token][creator] += creatorFee;
        operatorFees[token] += operatorFee;
        emit MarketsBetFeeCollected(marketCommitment, token, creator,
creatorFee, operatorFee);
        losingPotAmount -= (creatorFee + operatorFee);
    }
```

Above we see that the `creator` and `operator` fee are applied in real time whenever a bet is revealed. The result is that after the values are updated, the new fee percentages will be immediately applied to all revealed bets. This retroactively applies the updated fees to all bets even those for markets that closed well before the updated fees. To ensure fairness to all bettors, fees should be taken according to the percentage at the time of the bet.

Lines of Code

MarketsBase.sol#L300-L314

```
    if (losingPotAmount > 0) {
        uint256 currentlyAvailable = availableLosingPot[marketCommitment];
        require(currentlyAvailable >= losingPotAmount,
MarketsInvalidResult(marketCommitment, resultCommitment));
        availableLosingPot[marketCommitment] = currentlyAvailable -
losingPotAmount;

        // only charge fees on the losing pot, to discourage markets that
        // are heavily imbalanced. If the losing pot is small (because it's
        // a very unlikely result), then creator fees are also small
@>      uint256 creatorFee = (creatorFeeDecimal * losingPotAmount) /
FEE_DIVISOR;
@>      uint256 operatorFee = (operatorFeeDecimal * losingPotAmount) /
FEE_DIVISOR;
        creatorFees[token][creator] += creatorFee;
        operatorFees[token] += operatorFee;
        emit MarketsBetFeeCollected(marketCommitment, token, creator,
creatorFee, operatorFee);
        losingPotAmount -= (creatorFee + operatorFee);
    }
```

Recommendation

`creatorFeeDecimal` and `operatorFeeDecimal` should be cached upon market resolution and cached values should be read upon redemption rather than using the current values.

Remediation

Fixed in commit 40f6609. `creatorFeeDecimal` and `operatorFeeDecimal` are now cached in the `betState` when the bet is placed.