# UBET Audit Report

Reviewed by: 0x52

Report Prepared by: BlueJayAudits

Review Date(s):
12/18/23 - 12/21/23

## Scope

The ubet-contracts-v1 repo was reviewed at the commit hash 2766b47

NOTE: This was not a complete review of the codebase and was instead a review of the changes proposed in PR #69

## Summary of Findings

| Severity | # of findings |
|---|---|
| High | 2 |

## [H-01] MarketMaker.sol is vulnerable to inflation attacks

### Details

FundingMath.sol#L21-L37

```
function calcFunding(uint256 collateralAdded, uint256 totalShares, uint256
poolValue)
        internal
        pure
        returns (uint256 sharesMinted)
    {
        if (totalShares == 0) {
            // funding when LP pool is empty
            sharesMinted = collateralAdded;
        } else {
            // mint LP tokens proportional to how much value the new
investment
            // brings to the pool


            // Something is very wrong if poolValue has gone to zero
            if (poolValue == 0) revert FundingErrors.PoolValueZero();
            sharesMinted = (collateralAdded *
totalShares).ceilDiv(poolValue); <- @audit always rounds up
        }
    }
```

When adding liquidity the above lines are used to determine the number of shares to mint to the depositor. The use of `ceilDiv` in the `sharesMinted` calculation means that a user is guaranteed to receive at least 1 share when depositing. This allows the vault to become vulnerable to inflation attacks.

To execute this a malicious user would do as follows:

1) Deposit a single wei of liquidity

2) Donate a large amount of collateral to inflate `poolValue`

3) Buy a large number of a single outcome token to pull a large amount of funding from the `MarketFundingPool`

4) Make a large number of 1 wei deposits.

5) Each deposit mints 1 share which dilutes the liquidity share of `MarketMakerFunding`

6) After pool has closed withdraw all shares for a large profit.

## Lines of Code

FundingMath.sol#L35

## Recommendation

There are quite a few different ways to approach the solution. OZ recommends using a virtual offset.

## Remediation

Mitigated here. A virtual offset of 10,000 has been added to the pool. This makes the capital requirements significantly higher as well as the number of transactions to exploit this.

## [H-02] After migration, the old MarketFundingPool contract will fail to correctly distribute fees to new MarketFundingPool

### Details

FundingPool.sol#L135-L142

```
    function _beforeTokenTransfer(address from, address to, uint256 amount)
internal override {
        if (from != address(0)) {
            // LP tokens being transferred away from a funder - any fees that
            // have accumulated so far due to trading activity should be
given
            // to the original owner for the period of time he held the LP
            // tokens
            withdrawFees(from);
        }
```

`FundingPool#_beforeTokenTransfer` causes fees to be claimed when claiming new `MarketFundingPool` shares from the old `MarketFundingPool`. When receiving fees, they must be confirmed with the parent pool using `MarketMaker#_afterFeesWithdrawn`. When the shares are claimed for the new `MarketFundingPool` the fees are sent to the old `MarketFundingPool` but are never confirmed. When this happens the fees instead distributed across all shares causing loss of yield for the user claiming their shares.

### Lines of Code

FundingPool.sol#L135

MarketMaker.sol#L603-L607

### Recommendation

Implement `_afterFeesWithdrawn` on `MarketFundingPool` to call `MarketMaker#_afterFeesWithdrawn` when sending fees to the previous `MarketFundingPool`.

### Remediation

Mitigated here. `_afterFeesWithdrawn` is now implemented as recommended above on `ParentFundingPool` (inherited by `MarketFundingPool`).