UBET 1.2 Audit Report

Reviewed by: 0x52

Review Date(s):
6/12/24 – 6/14/24

Fix Review Date(s):
7/11/24

0x52 Background

As an independent smart contract auditor, I have completed over 100 separate reviews. I primarily compete in public contests as well as conducting private reviews (like this one here). I have more than 30 1st place finishes (and counting) in public contests on Code4rena and Sherlock I have also partnered with SpearbitDAO as a Lead Security researcher. My work has helped to secure over $1 billion in TVL across 100+ protocols.

Scope

The ubet-contract-v1 repo was reviewed at commit hash 6415782

As an update audit, only changes from commit c5eae81 to commit 6415782 were reviewed and other changes were considered out-of-scope.

Fixes were reviewed at commit hash 36cf1c3

Summary of Findings

Severity	# of findings
High	3
Medium	2

[H-01] Donations to MarketMaker will completely freeze all buy/sell capability of market

Details

```
    function getTargetBalance()
        public
        view
        returns (AmmMath.TargetContext memory targetContext, uint256[] memory
fairPriceDecimals)
    {
        // The logic is such that any excess collateral is always returned to the
parent
        uint256 localReserves = reserves();
        assert(localReserves == 0);
```

Inside getTargetBalance, an assert statement is used to make sure that all underlying tokens are either split to provide liquidity or returned back to the parent funding pool. While this is true in normal operations, this can be easily DOS'd by donating a single wei of underlying token.

```
    function buyFor(
        ...
    ) public returns (uint256 outcomeTokensBought, uint256 feeAmount, uint256[]
memory spontaneousPrices) {
        if (isHalted()) revert MarketHalted();
        if (investmentAmount < minInvestment) revert InvalidInvestmentAmount();

        uint256 tokensToMint;
        uint256 refundIndex;
        AmmMath.ParentOperations memory parentOps;
        {
            (AmmMath.TargetContext memory targetContext, uint256[] memory
fairPriceDecimals) = getTargetBalance();
            refundIndex = AmmMath.getRefundIndex(targetContext);
            (outcomeTokensBought, tokensToMint, feeAmount, spontaneousPrices,
parentOps) =
                _calcBuyAmount(investmentAmount, outcomeIndex, extraFeeDecimal,
targetContext, fairPriceDecimals);
        }
```

As seen above `buyFor` calls `getTargetBalance`. After donation, this will revert and cause all buy/sell capability to be broken, rendering the pool mostly useless.

This could be used under a variety of circumstances such as trying to force refunds to certain markets or block other users from buying or selling their choice.

Lines of Code

MarketMaker.sol#L672-L700

Recommendation

Remove the assert statement

Remediation

Fixed as recommended in commit ed00ebf.

[H-02] Frontrunning or reorg attacks can be used to corrupt initial pricing data to drain funds from MarketMaker

Details

```
    function prepareCondition(
        ...
    ) public returns (ConditionID) {
        // Limit of 256 because we use a partition array that is a number of 256
bits.
        if (outcomeSlotCount < 2 || outcomeSlotCount > 255) revert
InvalidOutcomeSlotsAmount();

        ConditionID conditionId = CTHelpers.getConditionId(conditionOracle,
questionId, outcomeSlotCount);

        // If not prepared, initialize, and emit the event, otherwise just return
existing conditionId
        if (payoutNumerators[conditionId].length == 0) {
            payoutNumerators[conditionId] = new uint256[](outcomeSlotCount);

            emit ConditionPreparation(conditionId, conditionOracle, questionId,
outcomeSlotCount);

            if (priceOracle != address(0x0)) {
                if (packedPrices.length > outcomeSlotCount * 2) revert
InvalidPrices();
                uint256 total = PackedPrices.sum(packedPrices);
                if (total != PackedPrices.DIVISOR) revert InvalidPrices();

                ConditionalTokensStorage.PriceStorage storage priceStorage =
priceStorage[conditionId];
                priceStorage.priceOracle = priceOracle;
                priceStorage.haltTime = haltTime_;
                priceStorage.packedPrices = packedPrices;

                …
            }
        }
        return conditionId;
    }
```

prepareCondition is a permissionless idempotent function used to initialize the condition, as well as setting initial pricing data and pricing oracle address. If the condition has already been prepared, it will simply return the conditionId.

Since oracle and pricing data is set via this function, it can be frontrun or can suffer reorgs which corrupt the price oracle or pricing data. This initial pricing corruption can be used purchase options at little to no cost causing financial damage to the liquidity pool.

Lines of Code

ConditionalTokens.sol#L92-L131

Recommendation

I would recommend 2 things:

1) Initial pricing data should be removed from prepareCondition and priceOracle should be used to derive conditionId

2) MarketFundingPool should call the priceOracle to set initial prices that way.

Remediation

Fixed in commit 52bb49f. Initial pricing data was removed from prepareCondition and prepareConditionByOracle was created to initialized pricing data in a permissioned way.

[H-03] Sending child shares before burning shares allow reentrancy vulnerability in ParentFundingPool#removeChildShares

Details

```
    function _afterTokenTransfer(address from, address to, uint256 amount) internal
override {
        // When address other than parent gets shares, immediately eject them to
        // maintain invariant that all funding is by parent
        if (from == getParentPool() && to != address(0x0)) {
            _removeFunding(to, amount);
        }
    }
```

When marketMaker shares are sent to an address other than the parent pool, _removeFunding is called to immediately burn the shares and return all conditional tokens to the user.

```
    function _removeFunding(address funder, uint256 sharesToBurn)
        private
        returns (uint256 collateral, uint256[] memory sendAmounts)
    {
        (collateral, sendAmounts) = _calcRemoveFunding(sharesToBurn);
        _burnSharesOf(funder, sharesToBurn);

        collateralToken.safeTransfer(funder, collateral);
        uint256 outcomeSlotCount = sendAmounts.length;
        conditionalTokens.safeBatchTransferFrom(
            address(this),
            funder,
            CTHelpers.getPositionIds(collateralToken, conditionId, outcomeSlotCount),
            sendAmounts,
            ""
        );
        ...
    }
```

We see above that tokens are transferred via safeBatchTransferFrom. This calls onERC1155BatchReceived on the receiver which allows them to gain control of contract execution.

```
    function removeChildShares(address child, uint256 sharesToBurn)
        external
        whenNotPaused
        returns (uint256 childSharesReturned, uint256 sharesBurnt)
    {
        ...

        if (childSharesReturned > 0) {
            assert(sharesBurnt > 0);
            // Update state
            funderShareRemovals[funder] = context.removals;
            totalChildValueLocked = context.totalChildValueLocked;
            valueHighPoint -= valueReturned;

            // Burn and transfer
            childPool.safeTransfer(funder, childSharesReturned);
            emit FundingRemovedAsToken(funder, uint256(bytes32(bytes20(child))),
childSharesReturned, sharesBurnt);
            _burnSharesOf(funder, sharesBurnt);
        }
```

ParentFundingPool#removeChildShares allows an LP to remove their parent shares as shares in a child pool. As shown above this immediately burns the child shares, allowing the receiver to gain execution the control. The problem here is that it is transferred in the middle of storage modification. totalChileValueLocked has been updating lowering the assets of the pool but shares have yet to be burned yet. This leads to a depressed share valuation. The attacker can use the gain in control to mint shares at low valuation. After the contract the valuation will no longer be depressed allowing the attacker to gain a large amount of assets and drain the pool.

Lines of Code

ParentFundingPool.sol#L259-L291

Recommendation

burnSharesOf should be called before child shares are transferred.

Remediation

removeChildShares fixed as recommended in commit 2b596e0. batchRemoveChildShares fixed in commit 70eb195 by rewriting function to be looped version of removeChildShares.

[M-01] Fees will be lost in the event that a user withdraws from MarketFundingPool and there isn't enough reserves to cover fees

Details

```
function _reevaluateGainsOnPool(bool force) private returns (uint256 fees) {
    uint256 lastBlock = lastFeeEvaluationBlock;
    uint256 period = feeEvaluationBlockPeriod;
    if (!force && (block.number - lastBlock <= period)) return fees;

    lastFeeEvaluationBlock = uint64(block.number);
    uint256 poolValue = getPoolValue();
    if (poolValue <= valueHighPoint) return fees;

    uint256 gains = poolValue - valueHighPoint;
    // Prevent taking more fees than available collateral
    fees = Math.min(reserves(), (gains * feePortion) / 256); <- fees minimized

    _retainFees(fees);
    valueHighPoint = poolValue - fees; <- fees that can't be paid are lost
}
```

When paying fees, they are set to the lower of fees owed or the amount of underlying asset. By setting valueHighPoint = poolValue - fees, the fees will be permanently lost if there is not enough reserves to cover the deposit. Assume our current valueHighPoint is 10,000 the pool takes a 1% fee, the current pool value is 20,000 and there is only $25 assets available. In this scenario the pool owes $100 (20,000 - 10,000 * 0.01) in fees but only has $25 to pay. When valueHighPoint is set it will set it to 19,975, which is the new value after the fee is taken. If the function is called again then fees will return as 0 and the other $75 are lost.

Lines of Code

ParentFundingPool.sol#L631-L646

Recommendation

Contract should revert on transactions if there are not enough fees and funds are removed.

Remediation

Fixed in commit 2b596e0. Transactions that remove value from the pool will revert if there is not enough collateral to cover fees. Transactions that add value when there is not enough fees will noop and valueHighPoint won't be updated.

[M-02] ParentFundingPool#removeCollateral fails to update valueHighPoint resulting in lost fees

Details

```
    function removeCollateral(uint256 sharesToBurn)
        external
        whenNotPaused
        returns (uint256 collateralReturned, uint256 sharesBurnt)
    {
        address funder = _msgSender();
        // force re-evaluation because some value is exiting
        _reevaluateGainsOnPool(true);

        (collateralReturned, sharesBurnt) = calcRemoveCollateral(funder,
sharesToBurn);
        if (sharesBurnt == 0) return (collateralReturned, sharesBurnt);

        funderShareRemovals[funder].removedAsCollateral += sharesBurnt;
        _burnSharesOf(funder, sharesBurnt);
        collateralToken.safeTransfer(funder, collateralReturned);

        uint256[] memory noTokens = new uint256[](0);
        emit FundingRemoved(funder, collateralReturned, noTokens, sharesBurnt);
    }
```

removeCollateral fails to properly update valueHighPoint resulting in it being too high. This results in fees failing to be properly collected on future gains.

Lines of Code

ParentFundingPool.sol#L233-L251

Recommendation

valueHighPoint should be reduced appropriately on withdrawals.

Remediation

Fixed as recommended in commit 2b596e0