TopCut Audit Report

Version 1.0

Mahdi Rostami

June 7, 2025

TopCut Audit Report

0xmahdirostami

June 7, 2025

Prepared by: Mahdi Rostami [Twitter]

Table of Contents

TopCut Security Review

A security review of the Possum-labs, TopCut smart contracts was done by mahdi rostami .
This audit report includes all the vulnerabilities, issues and code improvements found during the security review.

Disclaimer

“Audits are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.”

- Secureum

Impact

- **High** Issues that lead to the loss of user funds. Such issues include:
  - Direct theft of any user funds, whether at rest or in motion.
  - Long-term freezing of user funds. Theft or long term freezing of unclaimed yield or other assets.
  - Protocol insolvency
- **Medium** Issues that lead to an economic loss but do not lead to direct loss of on-chain assets. Examples are:
  - Gas griefing attacks (make users overpay for gas)
  - Attacks that make essential functionality of the contracts temporarily unusable or inaccessible.
  - Short-term freezing of user funds.
- **Low** Issues where the behavior of the contracts differs from the intended behavior (as described in the docs and by common sense), but no funds are at risk.

Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

Executive summary

Overview

Project Name	TopCut
Repository	Link
Commit Hash	ce88e7
Docs	No docs
Methods	Manual Review

Scope

File
TopCut

Compatibilities

- Solc Version: 0.8.19
- Chain(s) to deploy contract to: Arbitrum

Known Issues

None

Issues found

Severity	Count
High	0
Medium	4
Low	4
Informational	6

Findings

Medium Severity

[M-1] Incorrect Affiliate Points Redemption Logic Causes Reward Dilutio

Description: In the claimAffiliateReward function, the contract incorrectly adds affiliatePoints[_refID] (the total accumulated points) to totalRedeemedAP instead of _pointsRedeemed (the number of points being redeemed in the current transaction). This causes totalRedeemedAP to increase much faster than intended, leading to reward dilution for subsequent users or for users redeeming rewards in multiple transactions. As a result, the economic model of the affiliate rewards system is compromised.

Location: link

Code:

```
1      totalRedeemedAP = (totalRedeemedAP + affiliatePoints[_refID] <
           MAX_AP_REDEEMED)
2          ? totalRedeemedAP + affiliatePoints[_refID]
```

Mitigation: Update the logic to increment totalRedeemedAP by the actual number of points being redeemed in the current transaction:

```
1      totalRedeemedAP = (totalRedeemedAP + _pointsRedeemed <
           MAX_AP_REDEEMED)
2          ? totalRedeemedAP + _pointsRedeemed
```

Remark:

Fixed. commit

[M-2] Excessive Maximum Cohort Size May Cause Out-of-Gas Errors

Description: The current _maxCohortSize is set to 3300, which can lead to significant gas consumption. For example, a test with 3300 participants used at least 22,493,443 gas. Given that the Arbitrum block gas limit is 30,000,000, this could cause transactions to fail due to out-of-gas errors in production. The impact is high as both cohort users will lose money.

Location: link

Code:

```
        if (_maxCohortSize < 330 || _maxCohortSize > 3300) revert
            InvalidConstructor();
```

Mitigation: Reduce the maximum allowed cohort size to ensure gas usage stays within safe limits.

Remark: Fixed. commit

[M-3] Missing Sequencer Uptime Check When Using Chainlink Oracle on L2

Description: Using Chainlink on L2 chains like Arbitrum requires checking the sequencer's status to prevent stale or manipulated prices during sequencer downtime. Without this check, malicious actors could exploit periods when the sequencer is down, as oracle prices may appear fresh but are not reliable.

Location: link

Code:

```
    ///@dev Get the settlement price and timestamp from the oracle
    (, int256 price,, uint256 updatedAt,) = ORACLE.latestRoundData();


    ///@dev Ensure that the oracle price was updated after or at the
        settlement time
    if (updatedAt < settlementTime) revert StaleOraclePrice();


    ///@dev Sanity check to filter out a bad oracle feed (0 or negative
        )
    if (price < 1) revert BrokenOraclePrice();
```

Mitigation: Follow Chainlink's recommended approach link for L2 sequencer feeds to ensure the sequencer is up and a grace period has passed before using oracle data.

Remark: Fixed. commit & commit

[M-4] Incomplete Chainlink Oracle Data Validation Allows Stale or Invalid Prices

Description: The contract's current validation of Chainlink oracle data is insufficient. It only checks if updatedAt is greater than or equal to settlementTime and that price is positive. However, this approach does not ensure that the oracle data is fresh or finalized. Specifically, it does not verify that answeredInRound matches roundId, nor does it check if updatedAt is recent relative to block.timestamp. As a result, stale or incomplete price data could be processed, potentially leading to incorrect settlements or vulnerabilities.

Location: link

Code:

```
        ///@dev Get the settlement price and timestamp from the oracle
        (, int256 price,, uint256 updatedAt,) = ORACLE.latestRoundData
            ();


        ///@dev Ensure that the oracle price was updated after or at
            the settlement time
        if (updatedAt < settlementTime) revert StaleOraclePrice();


        ///@dev Sanity check to filter out a bad oracle feed (0 or
            negative)
        if (price < 1) revert BrokenOraclePrice();
```

Mitigation:

```
function getSecurePrice() external view returns (int256) {
    (
        uint80 roundId,
        int256 price,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    ) = priceFeed.latestRoundData();

    // Check for stale data
    require(answeredInRound >= roundId, "Stale price data");

    // Check for round completion
    require(updatedAt != 0, "Round not complete");

    // Check for price freshness (e.g., within last 3600 seconds)
    require(block.timestamp - updatedAt <= STALENESS_THRESHOLD, "Price
        too old");

    if (updatedAt < settlementTime) revert StaleOraclePrice();
```

```
20
21      // Check for valid price
22      require(price > 0, "Invalid price");
23
24      return price;
```

Remark: Fixed. commit & commit

Low Severity

[L-1] Inefficient Initial Settlement Timing

Description: In the first round, nextSettlement is set to a value greater than _tradeDuration * 2. This causes the settlement for the first cohort to occur at least at block.timestamp + _tradeDuration * 2 + _tradeDuration, introducing unnecessary delay.

Location: link

Code:

```
1          if (_firstSettlementTime < block.timestamp + _tradeDuration *
               2) revert InvalidConstructor();
```

Mitigation: Check _firstSettlementTime to be >= block.timestamp + _tradeDuration.

Remark: Fixed. commit

[L-2] Inefficient Claim Handling Prevents Partial Withdrawals

Description: The claim function attempts to transfer the entire claimable amount for a user in a single transaction. If the contract's balance is insufficient to cover the full amount, the transaction reverts, preventing the user from withdrawing any portion of their claim. This approach is inefficient, especially for users with large claim amounts who may prefer to withdraw available funds incrementally.

Location: link

Code:

```
1      uint256 amount = claimAmounts[user];
```

Mitigation: Allow users to specify the amount they wish to claim, enabling partial withdrawals when the contract balance cannot cover the full claim.

Remark: Acknowledged.

[L-3] Missing Cohort ID Input in castPrediction Can Lead to Incorrect Cohort Assignment

Description: We found that castPrediction does not take a cohortId as an input. As a result, if a transaction is delayed by the frontend or for other reasons, the prediction may be cast into an unexpected cohort.

Location: link

Mitigation: Require the cohort ID as an input to castPrediction and validate it to ensure predictions are assigned to the intended cohort.

Remark: Fixed. commit

[L-4] Insufficient Keeper Incentives for Small/Empty Cohort Settlement

Description: Currently, keepers receive rewards proportional to cohort size (_cohortSize * TRADE_SIZE * SHARE_KEEPER), creating a disincentive to process small or empty cohorts. This leads to timing issues as these cohorts may be neglected, potentially disrupting the protocol's expected settlement schedule and causing operational delays.

Location: link

Code:

```
        // INTERACTIONS
        ///@dev Compensate the keeper based on the number of traders in the Cohort
        uint256 keeperReward = (_cohortSize * TRADE_SIZE * SHARE_KEEPER) / SHARE_PRECISION;


        (bool sent,) = payable(msg.sender).call{value: keeperReward}("");
        if (!sent) revert FailedToSendKeeperReward();
```

Mitigation: Implement a minimum base reward for keepers regardless of cohort size to ensure all cohorts are processed in a timely manner, Replace the direct transfer mechanism with a claimable reward system to handle cases where contract funds are insufficient.

Remark: Fixed. commit

Info

1. The error `FailedToSkimSurplus();` is declared but not used in `TopCutMarket`.

2. In `castPrediction()`, it's preferable to compare `msg.value` with the immutable `TRADE_SIZE` and proceed using `TRADE_SIZE` instead of introducing a new `tradeSize` variable.
3. The variables `vaultReward` and `frontendReward` can be declared as `immutable` and calculated in the constructor for improved efficiency and clarity.
4. The comparison of `currentMaxValue` in `settleCohort` currently favors later users over the first user. To ensure the later user with the maximum value is selected, update the comparison from `diff > currentMaxValue` to `diff >= currentMaxValue`:

```
-     if (diff > currentMaxValue) {
+     if (diff >= currentMaxValue) {
      currentMaxValue = diff;
      currentMaxIndex = i;
      }
```

5. The error `FailedToSendNativeToken();` is declared but not used in `TopCutNFT`.
6. The `activeCohortID` variable is currently private, preventing users from viewing which cohort they are casting into; make it **`public`** to improve transparency.

Remark: Fixed. commit & commit