Outcome Market Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Hans

September 23, 2024

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

OutcomeMarket is a specialized smart contract system designed to create a prediction market for the 2024 US presidential election outcome. It allows users to mint conditional tokens using USDC as collateral, providing exposure to different election outcomes through various exchanges. The system is designed to be deployed on multiple blockchains, Ethereum, Base, and Arbitrum.

4.1 Actors and Roles

1. **Actors**:
    - **Users**: General users who can mint outcome tokens, trade them on exchanges, and redeem them for collateral.
    - **Oracle**: An external entity responsible for providing the accurate election result. The Oracle contract has an owner who can change the actual oracle to finalize the election result.
2. **Roles**:
    - The system is designed to be permissionless, with no specific roles or privileged accounts within the main contracts.

4.2 Key Components

1. **OutcomeMarket**: The main contract managing the minting, redemption, and resolution processes.
2. **OutcomeERC20**: Tokens representing different election outcomes (Trump and Harris).
3. **Collateral Token**: Native USDC (6 decimals) used as collateral across all supported chains.
4. **ElectionOracle**: Provided by Chaos Labs, responsible for delivering the election result.

4.3 Minting and Redemption Flow

1. **Minting**:
    - Users deposit USDC to mint an equal amount of tokens for each possible outcome (Trump and Harris).
    - Minting is only allowed before market resolution.
    - Minted tokens can be traded on exchanges for preferred exposure.
2. **Redemption**:
    - After market resolution, users can redeem outcome tokens for USDC.
    - If either Trump or Harris wins, the corresponding tokens are redeemable 1:1 for USDC.
    - If neither wins, both token types are redeemable at 0.5 USDC per token.

4.4 Resolution Flow

1. The `resolve()` function can be called by anyone after the election is finalized.
2. It checks with the oracle to determine if the election is finalized and retrieves the result.
3. Based on the oracle's response, it sets the winning outcome or handles the case where neither candidate won.

4.5 Multi-chain Deployment

The system is designed to be deployed on Ethereum, Base, and Arbitrum, using the native USDC on each chain as collateral.

4.6 General Security Considerations

1. **Oracle Dependence**: The system relies on Chaos Labs for the election result.
2. **MEV Opportunities**: The system acknowledges potential MEV opportunities during resolution but considers this acceptable due to the expected delay in oracle result delivery.
3. **Cross-chain Risks**: Deployment across multiple chains may introduce complexities in maintaining consistent state and handling potential chain-specific issues.

5 Audit Scope

```
OutcomeERC20.sol
OutcomeMarket.sol
ElectionOracle.sol
```

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Outcome Market smart contracts provided by Wintermute. In this period, a total of 4 issues were found.

`OutcomeMarket` and `OutcomeERC20` are provided by Wintermute Research, and `ElectionOracle` is by Chaos Labs. The oracle provider will do all necessary work to ensure that the right result will be provided to their contract according to the methodology.

Summary

Project Name	Outcome Market
Repository	Outcome-Market
Commit	3573cc1d58ac...
Audit Timeline	Sep 19th - Sep 20th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Informational	1
Gas Optimizations	0
Total Issues	4

Summary of Findings

[L-1] Dummy finalization by the election oracle is possible due to insufficient validation	Resolved
[L-2] Unnecessary rounding in `_handleRedeemCaseOther`	Resolved
[L-3] Funds will be frozen if the oracle fails to provide the result	Acknowledged
[I-1] Minor typo in comments	Resolved

7 Findings

7.1 Low Risk

7.1.1 Dummy finalization by the election oracle is possible due to insufficient validation

Description: `OutcomeMarket` relies on the `ElectionOracle` to decide the election status and result using the functions `ElectionOracle::isElectionFinalized()` and `ElectionOracle::getElectionResult()`.

The function `isElectionFinalized()` is a wrapper of a state `isResultFinalized` and it is supposed to return True only when the election status is finalized. This state is set to true in the function `ElectionOracle::finalizeElectionResult()` that is implemented as below.

```
ElectionOracle.sol
50:     function finalizeElectionResult(ElectionResult _finalResult) external onlyRole(ORACLE_ROLE) {
51:         require(block.timestamp >= minEndOfElectionTimestamp, "Cannot finalize before the end of
↪  the election period.");
52:         require(result == ElectionResult.NotSet, "Election result is already finalized.");
53:
54:         result = _finalResult;
55:         isResultFinalized = true;
56:         resultFinalizationTimestamp = block.timestamp;
57:
58:         emit ElectionFinalized(_finalResult, block.timestamp);
59:     }
```

But this function does not validate if the input `_finalResult` is a valid election result and if the oracle calls the function with `_finalResult=ElectionOracle.ElectionResult.NotSet`, it goes through the logic and `isResultFinalized` is set to true.

Impact: Because the oracle is a trusted actor, the likelihood is very low. Furthermore, the dummy finalization does not affect the `OutcomeMarket` because `OutcomeMarket::_handleOracleAnswer` reverts for the `NotSet` state. Overall, the impact is LOW.

Proof Of Concept: Below is a test function that proves the issue. (Put in the `ElectionOracleTest`.)

```
    function testAuditDummyFinalization() public {
        vm.startPrank(oracle);
        vm.warp(minEndOfElectionTimestamp + 1);

        electionOracle.finalizeElectionResult(ElectionOracle.ElectionResult.NotSet); //dummy
        ↪  finalization

        assertEq(electionOracle.isElectionFinalized(), true);
        assertEq(uint8(electionOracle.getElectionResult()),
        ↪  uint8(ElectionOracle.ElectionResult.NotSet));

        electionOracle.finalizeElectionResult(ElectionOracle.ElectionResult.Trump); //finalize again
        vm.stopPrank();
    }
```

Recommended Mitigation: Add a validation for the `_finalResult` in the function `ElectionOracle::finalizeElectionResult()`.

```
require(_finalResult != ElectionResult.NotSet, "Invalid election result is provided.");
```

Client: Fixed in PR: CYF001

Cyfrin: Verified.

7.1.2 Unnecessary rounding in `_handleRedeemCaseOther`

Description: In the case where neither Trump nor Harris won, the outcome tokens are redeemed by the function `OutcomeMarket::_handleRedeemCaseOther()`.

```
OutcomeMarket.sol
141:     function _handleRedeemCaseOther() internal {
142:         uint256 _totalCollateral;
143:         uint256 _outcomeTokensTotalSupply = outcomeTokens[0].totalSupply() +
↪  outcomeTokens[1].totalSupply();
144:         for (uint256 i = 0; i < outcomeTokens.length; i++) {
145:             OutcomeERC20 _outcomeToken = outcomeTokens[i];
146:             uint256 _senderBalance = _outcomeToken.balanceOf(msg.sender);
147:             if (_senderBalance > 0) {
148:                 // Note: exchange rate in this case is 1 outcome token -> 0.5 collateral token
149:                 uint256 _collateralAmount = _calcCollateralShare(_senderBalance,
↪  _outcomeTokensTotalSupply);
150:                 _totalCollateral += _collateralAmount;
151:                 _outcomeToken.burn(msg.sender, _senderBalance);
152:             }
153:         }
154:         if (_totalCollateral != 0) {
155:             collateralToken.transfer(msg.sender, _totalCollateral);
156:             emit PayoutDistributed(msg.sender, _totalCollateral);
157:         } else {
158:             revert OutcomeMarket__NothingToRedeem();
159:         }
160:     }
```

Looking at the implementation, the collateral amount for each token is calculated separately and summed. (L149) The function `_calcCollateralShare` includes a call to `Math.mulDiv` which will cause rounding down. So the curren implementation includes two rounding down in the process and this would cause unnecessary rounding in the overall result.

Impact: The impact is very LOW because the amount difference would be tiny and neglectable.

Proof Of Concept: Below is a test case in Foundry and it shows an extreme case where a user can not redeem his outcome token due to rounding down.

```
    function testAuditLossInRedeemForCaseOther() external {
        address alice = address(0x1);
        uint amount = 1; // 1e-6 USDC
        _mint(alice, amount, false);
        _resolveOracle(ElectionResult.Other);

        vm.prank(alice);
        vm.expectRevert(OutcomeMarket.OutcomeMarket__NothingToRedeem.selector);
        market.redeem();
    }
```

Recommended Mitigation: Please consider revising the function `_handleRedeemCaseOther` so that the rounding happens only once.

```
_totalCollateral = _calcCollateralShare(outcomeTokens[0].balanceOf(msg.sender) +
↪  outcomeTokens[1].balanceOf(msg.sender), _outcomeTokensTotalSupply);
```

Client: Fixed in PR: CYF002

Cyfrin: Verified.

7.1.3 Funds will be frozen if the oracle fails to provide the result

Description: The `OutcomeMarket` allows redemption only when the election result is settled by the oracle. If the oracle fails to finalize the election result, funds will be frozen. While we acknowledge that the oracle is a very critical part that is trusted, it would be better to have a backup mechanism for additional safety.

Impact: Collateral funds (USDC) could be frozen if the oracle fails the result.

Recommended Mitigation: Implement a time-based fallback for resolution in case of oracle failure.

Client: Acknowledged. Based on our discussions with Chaos Labs, that's an extremely tail risk, and we'll keep this code as is, at least for this specific market.

Cyfrin: Acknowledged.

7.2 Informational

7.2.1 Minor typo in comments

Description: There is a minor typo in the comments of `OutcomeMarket`.

```
OutcomeMarket.sol
40:  /// @notice Emitted when a user creates a new positions
```

Client: Fixed in PR: WMUT-13 and CYF004

Cyfrin: Verified.