O.Lab

Audit Report

Mon Nov 11 2024

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

O.Lab Audit Report

1 Executive Summary

1.1 Project Information

Description	The O.Lab CTF Exchange is an exchange protocol that facilitates atomic swaps between Conditional Tokens Framework ERC1155 assets and an ERC20 collateral asset
Type	DeFi
Auditors	ScaleBit
Timeline	Wed Oct 09 2024 - Mon Nov 11 2024
Languages	Solidity
Platform	BSC
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/OpinionLabs/prediction-market-smart-contract-v2
Commits	7735d82736955d8e8fde38a9981c67cb8345d4f6

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BEX	ctf-exchange/src/exchange/BaseExchange.sol	e0cf90fc984237ddf7dc6bf8282dc00459fba55c
OST	ctf-exchange/src/exchange/libraries/OrderStructs.sol	c725f8e0c47b682f3848b7c132081df11485facf
OLABSL	ctf-exchange/src/exchange/libraries/OLABSafeLib.sol	07c3463e58fe9c896bde77e5fedc43d0f281d004
CHE	ctf-exchange/src/exchange/libraries/CalculatorHelper.sol	24db647d321ec7ccb98ea7828c410c35b6bea4d7
THE1	ctf-exchange/src/exchange/libraries/TransferHelper.sol	943d67899c1f7f208ab33c8a6b2eb88cac95dd43
OLABPL	ctf-exchange/src/exchange/libraries/OLABProxyLib.sol	e245ab584e0fb3ffa1ef9ef54599091f0cdd8b6d
TRA	ctf-exchange/src/exchange/mixins/Trading.sol	684d13f230ee3f0b067e69e6590102af9d6f78b8
OLABFH	ctf-exchange/src/exchange/mixins/OLABFactoryHelper.sol	ceaaa9e055c465c9ce3f0b36e0c7388ee769480d
REG	ctf-exchange/src/exchange/mixins/Registry.sol	1fdcc435c425c3b62833e0c5d26206b5e4db6e8c
AOP	ctf-exchange/src/exchange/mixins/AssetOperations.sol	158d0b544ebe1402b42ed0a4b9df2a7ac4758cb5
ASS	ctf-exchange/src/exchange/mixins/Assets.sol	944dbc20dbea265a9f8bcb4655188b5b27038a1c

SIG	ctf-exchange/src/exchange/mixins/Signatures.sol	d3cad6b92797444799726efc73a619f480be7abe
AUT1	ctf-exchange/src/exchange/mixins/Auth.sol	ceb3e5297657bef3f92fa187f8989df45859b318
NMA	ctf-exchange/src/exchange/mixins/NonceManager.sol	70ad6c00164ddc647eacb92df803f6799d0f418b
PAU	ctf-exchange/src/exchange/mixins/Pausable.sol	2826bcafe42b143e4256063b9457131e9aeec197
FEE	ctf-exchange/src/exchange/mixins/Fees.sol	be16867c573dfb1668285b4c118f1abe1baedcee
HAS	ctf-exchange/src/exchange/mixins/Hashing.sol	5eaaaa2365d0b603268efbc024896d3fde14b4a1
CTFE	ctf-exchange/src/exchange/CTFExchange.sol	e867efc1cd30feafd4c49db963fe6f15cfb1a071
CTO	conditional-tokens-contracts/contracts/ConditionalTokens.sol	8caacb722b80fd0dac21221b3ce9defcd15183a1

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	5	3
Informational	2	1	1
Minor	4	2	2
Medium	0	0	0
Major	2	2	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by O.Lab to identify any potential issues and vulnerabilities in the source code of the CTF Exchange smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
CHE-1	Zero-Fee Orders May Lead to Revenue Loss and Potential Abuse	Minor	Fixed
CTO-1	Use `safeTransferFrom()` instead of `transferFrom()`	Major	Fixed
CTO-2	The Protocol does not Support Fee-On-Transfer Tokens	Informational	Acknowledged
CTO-3	The `collateralToken` Lacks Validation	Informational	Fixed
TRA-1	Draining the Order Maker's Funds without Providing any Tokens in return	Major	Fixed
TRA-2	The Operator Executing Invalid Order Fills can Result in Financial Losses	Minor	Fixed
TRA-3	Simplify Sending Fees	Minor	Acknowledged
TRA-4	Unable To Return Unexpected Tokens	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the CTF Exchange Smart Contract :

Admin

- The admin can set the treasury through the `setTreasury` function.
- The admin can pause trading through the `pauseTrading` function.
- The admin can unpause trading through the `unpauseTrading` function.
- The admin can set a new Proxy Wallet factory through the `setProxyFactory` function.
- The admin can set a new safe factory through the `setSafeFactory` function.

Operator

- The operator can fill an order through the `fillOrder` function.
- The operator can fill multiple orders through the `fillOrders` function.
- The operator can match a taker order against a list of maker orders through the `matchOrders` function.
- The operator can register a tokenId, its complement, and its conditionId for trading through the `registerToken` function.

User

- The uses can cancel an order through the `cancelOrder` function.
- The uses can cancel a set of orders through the `cancelOrders` function.

4 Findings

CHE-1 Zero-Fee Orders May Lead to Revenue Loss and Potential Abuse

Severity: Minor

Status: Fixed

Code Location:

ctf-exchange/src/exchange/libraries/CalculatorHelper.sol#27-49

Descriptions:

In the current implementation of the `calculateFee` function, if an order's `feeRateBps` is set to 0, no fees will be charged. This design allows for the creation of completely free transactions, which may result in extensive use of zero-fee orders potentially significantly reducing the protocol's revenue. Even though no fees are charged, processing these orders still consumes system resources.

```
    function calculateFee(
        uint256 feeRateBps,
        uint256 outcomeTokens,
        uint256 makerAmount,
        uint256 takerAmount,
        Side side
    ) internal pure returns (uint256 fee) {
        if (feeRateBps > 0) {
            uint256 price = _calculatePrice(makerAmount, takerAmount, side);
            if (price > 0 && price <= ONE) {
                if (side == Side.BUY) {
                    // Fee charged on Token Proceeds:
                    // baseRate * min(price, 1-price) * (outcomeTokens/price)
                    fee = (feeRateBps * min(price, ONE - price) * outcomeTokens) / (price *
BPS_DIVISOR);
                } else {
                    // Fee charged on Collateral proceeds:
                    // baseRate * min(price, 1-price) * outcomeTokens
                    fee = feeRateBps * min(price, ONE - price) * outcomeTokens / (BPS_DIVISOR *
ONE);
                }
```

```
        }
    }
}
```

Suggestion:

It is recommended to set a minimum fee rate.

Resolution:

This issue has been fixed. The client has added an off-chain verification mechanism.

CTO-1 Use `safeTransferFrom()` instead of `transferFrom()`

Severity: Major

Status: Fixed

Code Location:

conditional-tokens-contracts/contracts/ConditionalTokens.sol#135

Descriptions:

In the `splitPosition()` function, if `parentCollectionId == bytes32(0)`, the protocol calls `collateralToken.transferFrom()` to transfer a certain amount of collateral tokens from `msg.sender` and uses `require` to check if the returned result is true.

```
        // Partitioning the full set of outcomes for the condition in this branch
        if (parentCollectionId == bytes32(0)) {
            require(collateralToken.transferFrom(msg.sender, address(this), amount),
"could not receive collateral tokens");
        }
```

However, some tokens, like USDT, do not return a value, which will cause the transaction to revert. https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code

```
    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function transferFrom(address _from, address _to, uint _value) public whenNotPaused
{
        require(!isBlackListed[_from]);
        if (deprecated) {
            return
UpgradedStandardToken(upgradedAddress).transferFromByLegacy(msg.sender, _from,
_to, _value);
        } else {
            return super.transferFrom(_from, _to, _value);
        }
    }
```

Suggestion:

It is recommended to replace `transferFrom()` with `safeTransferFrom()` to handle tokens that do not return a value properly.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

CTO-2 The Protocol does not Support Fee-On-Transfer Tokens

Severity: Informational

Status: Acknowledged

Code Location:

conditional-tokens-contracts/contracts/ConditionalTokens.sol#135

Descriptions:

In the `splitPosition()` function, the protocol calls `collateralToken.transferFrom()` to transfer a certain amount of tokens from `msg.sender` and then calls `_batchMint()` to mint the corresponding amount of tokens to `msg.sender`.

```
if (freeIndexSet == 0) {
        // Partitioning the full set of outcomes for the condition in this branch
        if (parentCollectionId == bytes32(0)) {
            require(collateralToken.transferFrom(msg.sender, address(this), amount),
"could not receive collateral tokens");
        } else {
            _burn(
                msg.sender,
                CTHelpers.getPositionId(collateralToken, parentCollectionId),
                amount
            );
        }
    } else {
        // Partitioning a subset of outcomes for the condition in this branch.
        // For example, for a condition with three outcomes A, B, and C, this branch
        // allows the splitting of a position $:(A|C) to positions $:(A) and $:(C).
        _burn(
            msg.sender,
            CTHelpers.getPositionId(collateralToken,
                CTHelpers.getCollectionId(parentCollectionId, conditionId, fullIndexSet ^
freeIndexSet)),
            amount
        );
    }

    _batchMint(
        msg.sender,
```

```
        // position ID is the ERC 1155 token ID
        positionIds,
        amounts,
        ""
    );
```

If the `collateralToken` is a fee-on-transfer token, the protocol will receive fewer tokens than expected, but it will mint more tokens to the user than it should.

Suggestion:

It is recommended to calculate the actual amount received by subtracting the protocol's balance after the transfer from its balance before the transfer.

CTO-3 The `collateralToken` Lacks Validation

Severity: Informational

Status: Fixed

Code Location:

conditional-tokens-contracts/contracts/ConditionalTokens.sol#106

Descriptions:

In the `splitPosition()` function, the protocol does not validate the `collateralToken`, allowing users to mint NFTs using any arbitrary ERC20 token they create.

```
function splitPosition(
    IERC20 collateralToken,
    bytes32 parentCollectionId,
    bytes32 conditionId,
    uint[] calldata partition,
    uint amount
) external {
    require(partition.length > 1, "got empty or singleton partition");
    uint outcomeSlotCount = payoutNumerators[conditionId].length;
    require(outcomeSlotCount > 0, "condition not prepared yet");
```

Suggestion:

It is recommended to verify that the `collateralToken` used matches the `collateralToken` in the CTF exchange.

Resolution:

This issue has been fixed. The client has added the following check:

```
require(conditionCollateral[conditionId] == address(collateralToken), "collateral token mismatch");
```

TRA-1 Draining the Order Maker's Funds without Providing any Tokens in return

Severity: Major

Status: Fixed

Code Location:

ctf-exchange/src/exchange/mixins/Trading.sol#89-109

Descriptions:

In the `_fillOrder()` function, the protocol first transfers funds, deducting the fee, from `msg.sender` to the order maker, and then transfers funds from the order maker to `msg.sender`.

```
        // Transfer order proceeds minus fees from msg.sender to order maker
        _transfer(msg.sender, order.maker, takerAssetId, taking - fee);

        // Transfer makingAmount from order maker to `to`
        _transfer(order.maker, to, makerAssetId, making);
```

The calculation for `takingAmount` is as follows:

```
takingAmount = makingAmount * takerAmount / makerAmount
```

Here:

- `makerAmount` is the maximum amount of tokens to be sold by the maker.
- `takerAmount` is the minimum amount of tokens to be received by the maker.

Let's assume the order maker wants to exchange 1e18 tokens for 3000 *1e6 tokens. Thus:

- `makerAmount = 1e18`
- `takerAmount = 3000 * 1e6`

The `takingAmount` calculation becomes:

```
takingAmount = makingAmount * (3000 * 1e6) / 1e18
```

Now, if `makingAmount` is less than `1e18 / (3000 * 1e6) = 333333333`, the calculated `takingAmount` will be zero. Since the fee is also based on the `takingAmount`, it will be zero as well. As a result, the order maker will receive 0 tokens from `msg.sender`, but will transfer 333333333 wei token to `msg.sender`.

A malicious attacker can exploit this by initiating multiple transactions with such small values, effectively draining the order maker's funds without providing any tokens in return.

Suggestion:

It is recommended to set a minimum `fillAmount` for users or to check that `takingAmount` is greater than 0.

Resolution:

This issue has been fixed, and the client has removed the `fillOrder` function.

TRA-2 The Operator Executing Invalid Order Fills can Result in Financial Losses

Severity: Minor

Status: Fixed

Code Location:

ctf-exchange/src/exchange/mixins/Trading.sol#89-109

Descriptions:

In the `_fillOrder()` function, the protocol first transfers funds, deducting the fee, from `msg.sender` to the order maker, and then transfers the `fillAmount` from the order maker to `msg.sender`.

```
  function _fillOrder(Order memory order, uint256 fillAmount, address to) internal {
    uint256 making = fillAmount;
    (uint256 taking, bytes32 orderHash) = _performOrderChecks(order, making);

    uint256 fee = CalculatorHelper.calculateFee(
      order.feeRateBps, order.side == Side.BUY ? taking : making, order.makerAmount,
order.takerAmount, order.side
    );

    (uint256 makerAssetId, uint256 takerAssetId) = _deriveAssetIds(order);

    // Transfer order proceeds minus fees from msg.sender to order maker
    _transfer(msg.sender, order.maker, takerAssetId, taking - fee);

    // Transfer makingAmount from order maker to `to`
    _transfer(order.maker, to, makerAssetId, making);

    // NOTE: Fees are "collected" by the Operator implicitly,
    // since the fee is deducted from the assets paid by the Operator

    emit OrderFilled(orderHash, order.maker, msg.sender, makerAssetId, takerAssetId,
making, taking, fee);
  }
```

However, the protocol does not check whether `fillAmount > 0`. Since the person executing the `fillOrder` is the operator, a malicious attacker can create multiple requests with a `fillAmount` of 0, forcing the operator to execute these transactions. This leads to the operator spending gas without any meaningful transaction, resulting in financial losses for the operator due to the wasted gas fees.

Suggestion:

It is recommended to check that `fillAmount > 0`.

Resolution:

This issue has been fixed, and the client will handle the validation off-chain.

TRA-3 Simplify Sending Fees

Severity: Minor

Status: Acknowledged

Code Location:

ctf-exchange/src/exchange/mixins/Trading.sol#242-273

Descriptions:

`Trading._fillFacingExchange` now transfers the fee from the contract to the operator on every call. When multiple maker orders are processed, there is a fee transfer for every one of them. The fee could be sent after all orders have been processed instead.

Suggestion:

It is recommended to send the fee after all orders have been processed.

TRA-4 Unable To Return Unexpected Tokens

Severity: Minor

Status: Acknowledged

Code Location:

ctf-exchange/src/exchange/mixins/Trading.sol#371-375

Descriptions:

Tokens that have been accidentally sent to the contract can not be recovered. Furthermore, if either the collateral token or one of the outcome tokens has been accidentally sent to the contract, the next executed taker order will receive these tokens due to the implementation of `Trading._updateTakingWithSurplus`.

```
  function _updateTakingWithSurplus(uint256 minimumAmount, uint256 tokenId)
internal returns (uint256) {
      uint256 actualAmount = _getBalance(tokenId);
      if (actualAmount < minimumAmount) revert TooLittleTokensReceived();
      return actualAmount;
  }
```

Suggestion:

It is recommended to add a function to return the user's token.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.