Protocol Audit Report

Version 1.0

Tim Sigl

September 5, 2024

Protocol Audit Report

Tim Sigl

2024/08

Prepared by: Lead Security Researcher Tim Sigl

Table of Contents

Protocol Summary

Tadle is a cutting-edge pre-market infrastructure designed to unlock illiquid assets in the crypto pre-market.

Our first product, the Points Marketplace, empowers projects to unlock the liquidity and value of points systems before conducting the Token Generation Event (TGE). By facilitating seamless trading and providing a secure, trustless environment, Tadle ensures that your community can engage with your tokens and points dynamically and efficiently.

Contest Summary

Sponsor: Tadle

Dates: Aug 5th, 2024 - Aug 12th, 2024

See more contest details here

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond with the following commit hash:

```
04fd8634701697184a3f3a5558b41c109866e5f8
```

Scope

1.229 nSLOC

```
src
|-- core
|   |-- CapitalPool.sol
|   |-- DeliveryPlace.sol
|   |-- PreMarkets.sol
|   |-- SystemConfig.sol
|   +-- TokenManager.sol
|-- factory
|   |-- ITadleFactory.sol
|   +-- TadleFactory.sol
|-- interfaces
|   |-- ICapitalPool.sol
|   |-- IDeliveryPlace.sol
|   |-- IPerMarkets.sol
|   |-- ISystemConfig.sol
|   +-- ITokenManager.sol
|-- libraries
|   |-- MarketPlaceLibraries.sol
|   +-- OfferLibraries.sol
+-- storage
    |-- CapitalPoolStorage.sol
    |-- DeliveryPlaceStorage.sol
    |-- OfferStatus.sol
    |-- PerMarketsStorage.sol
    |-- SystemConfigStorage.sol
    +-- TokenManagerStorage.sol
```

Roles

- Maker
  - Create buy offer
  - Create sell offer
  - Cancel your offer
  - Abort your offer
- Taker
  - Place taker orders
  - Relist stocks as new offers
- Sell Offer Maker
  - Deliver tokens during settlement

- General User
  - Fetch balances info
  - Withdraw funds from your balances
- Admin (Trust)
  - Create a marketplace
  - Take a marketplace offline
  - Initialize system parameters, like WETH contract address, referral commission rate, etc.
  - Set up collateral token list, like ETH, USDC, LINK, ankrETH, etc.
  - Set `TGE` parameters for settlement, like token contract address, TGE time, etc.
  - Grant privileges for users' commission rates
  - Pause all the markets

Issues found

Severity	Number of Findings
High	2
Medium	0
Low	0
Info	0
Total	0

Findings

High

H-01. Insufficient Allowance Prevents User Withdrawals

Summary The `TokenManager` contract fails to properly manage allowances when withdrawing tokens from the `CapitalPool`, leading to locked user funds.

Vulnerability Details The vulnerability exists in the `TokenManager::withdraw` function. When a user attempts to withdraw their tokens, the function tries to transfer tokens directly from the `CapitalPool` to the user without ensuring proper allowances are set.

Specifically:

1. In the withdraw function (line 137-189), the contract attempts to transfer tokens using `_safe_transfer_from`:

```
_safe_transfer_from(_tokenAddress, capitalPoolAddr, _msgSender(),
    claimAbleAmount);
```

1. This call fails because the TokenManager does not have sufficient allowance to transfer tokens on behalf of the CapitalPool.
2. The contract does have a mechanism to check and set allowances in the _transfer function (lines 233-262), but this is not utilized in the withdraw function.

Proof of code

Insert the following code snippet into `PreMarkets.t.sol`. It will revert the transaction due to insufficient allowance:

```
function testInsufficientAllowanceWhenWithdrawing() public {
        address testSeller = makeAddr("testSeller");
        address testBuyer = makeAddr("testBuyer");
        deal(address(mockPointToken), testSeller, 10 ether);
        deal(address(mockUSDCToken), testSeller, 12e15);
        deal(address(mockUSDCToken), testBuyer, 1 ether);

        vm.startPrank(testSeller);
        mockUSDCToken.approve(address(tokenManager), type(uint256).max)
            ;
        mockPointToken.approve(address(tokenManager), type(uint256).max
            );

        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Ask, // Sell points
                OfferSettleType.Turbo
            )
        );
        vm.stopPrank();

        vm.startPrank(testBuyer);
        mockUSDCToken.approve(address(tokenManager), type(uint256).max)
            ;
        address offerAddr = GenerateAddress.generateOfferAddress(0);
```

```
29          preMarktes.createTaker(offerAddr, 1000);
30          vm.stopPrank();
31
32          vm.prank(user1);
33          systemConfig.updateMarket(
34              "Backpack",
35              address(mockPointToken),
36              0.01 * 1e18,
37              block.timestamp - 1, // This sets the TGE to the past so
                    MarketPlaceStatus will be AskSettling and no new offers
                    can be created
38              3600
39          );
40
41          vm.startPrank(testSeller);
42          mockUSDCToken.approve(address(tokenManager), type(uint256).max)
                ;
43          mockPointToken.approve(address(tokenManager), type(uint256).max
                );
44          deliveryPlace.settleAskMaker(offerAddr, 1000);
45          // The testSeller wants to withdraw the revenue from selling
                1000 points to testBuyer
46          tokenManager.withdraw(address(mockUSDCToken), TokenBalanceType.
                SalesRevenue);
47          vm.stopPrank();
48      }
```

Impact Users are unable to withdraw their rightful tokens, effectively locking their funds in the contract.

Tools Used

- Manual code review
- Forge unit tests

Recommendations

1. Modify the withdraw function to approve the `TokenManager` to transfer tokens on the behalf of the `CapitalPool` before transferring tokens to the user.

```
1  function withdraw(
2      address _tokenAddress,
3      TokenBalanceType _tokenBalanceType
4  ) external whenNotPaused {
5      // ... (existing code)
6
```

```
    if (_tokenAddress == wrappedNativeToken) {
        // ... (existing native token handling)
    } else {
+       ICapitalPool(capitalPoolAddr).approve(_tokenAddress);
            _safe_transfer_from(
                _tokenAddress,
                capitalPoolAddr,
                _msgSender(),
                claimAbleAmount
            );
    }

    // ... (remaining code)
}
```

H-02. `DeliveryPlace::closeBidTaker` Adds Wrong Token Balance to Taker Preventing Withdrawal of Point Tokens

Summary After settlement, the taker of an ask offer should receive the point tokens they bought. This is done through a token balance which is added to the taker's account in the `TokenManager::addTokenBalance` function. The system incorrectly adds a balance of the collateral token (USDC in the following PoC) to the taker instead of point tokens which the actually bought.

Vulnerability Details When a taker (buyer) purchases point tokens from a maker (seller), the following process occurs:

1. The maker creates an offer to sell point tokens (`PreMarkets::createOffer`)
2. The taker fills the offer (`PreMarkets::createTaker`)
3. The owner updates the market to after the TGE (`SystemConfig::updateMarket`)
4. The maker settles the offer (`DeliveryPlace::settleAskMaker`)
5. The taker closes the bid taker (`DeliveryPlace::closeBidTaker`)

In the last step a token balance is added to the taker's account which allows them to withdraw the point tokens they bought later on (`DeliveryPlace` line 195-200):

```
    // (other existing code)
    tokenManager.addTokenBalance(
        TokenBalanceType.PointToken,
        _msgSender(),
        makerInfo.tokenAddress,
        pointTokenAmount
    );
```

However, `makerInfo.tokenAddress` is not the address of the point token but the collateral token. Therefore the `TokenManager::addTokenBalance` function give the taker the wrong token balance to withdraw (`TokenManager` line 113-129):

```
function addTokenBalance(
        TokenBalanceType _tokenBalanceType,
        address _accountAddress,
        address _tokenAddress, // This should be the point token
            address but is the collateral token address
        uint256 _amount
    ) external onlyRelatedContracts(tadleFactory, _msgSender()) {
        userTokenBalanceMap[_accountAddress][_tokenAddress][
            _tokenBalanceType
        ] += _amount;

        emit AddTokenBalance(
            _accountAddress,
            _tokenAddress,
            _tokenBalanceType,
            _amount
        );
    }
```

Overview

The following PoC demonstrates the issue. The test will revert because the expected event (allow taker to withdraw 1000 point tokens) does not match the emitted event (allow taker to withdraw 1000 collateral tokens (USDC)).

Actors

- **testSeller**: The maker who wants to sell point token
- **testBuyer**: The taker who wants to buy point token
- **user1**: The owner of the market

Please copy the test into `PreMarkets.t.sol`:

```
    event AddTokenBalance(
        address indexed accountAddress,
        address indexed tokenAddress,
        TokenBalanceType indexed tokenBalanceType,
        uint256 amount
    );

    function testWrongAddTokenBalance() public {
        address testSeller = makeAddr("testSeller");
        address testBuyer = makeAddr("testBuyer");
        // Give maker (seller) the point tokens they want to sell and
            the collateral
```

```
12          deal(address(mockPointToken), testSeller, 1e19);
13          deal(address(mockUSDCToken), testSeller, 12e15);
14          // Give taker (buyer) the tokens needed to buy the points
15          deal(address(mockUSDCToken), testBuyer, 2e18);
16
17          // Maker creates offer for 1000 points
18          vm.startPrank(testSeller);
19          mockUSDCToken.approve(address(tokenManager), type(uint256).max)
                ;
20          mockPointToken.approve(address(tokenManager), type(uint256).max
                );
21
22          preMarktes.createOffer(
23              CreateOfferParams(
24                  marketPlace,
25                  address(mockUSDCToken),
26                  1000,
27                  0.01 * 1e18,
28                  12000,
29                  300,
30                  OfferType.Ask, // Sell points
31                  OfferSettleType.Turbo
32              )
33          );
34          vm.stopPrank();
35
36          // Taker fulfills the offer
37          vm.startPrank(testBuyer);
38          // Approve tokenManager to take the purchase price
39          mockUSDCToken.approve(address(tokenManager), type(uint256).max)
                ;
40          address offerAddr = GenerateAddress.generateOfferAddress(0);
41          preMarktes.createTaker(offerAddr, 1000);
42          vm.stopPrank();
43
44          // Owner updates the market to after TGE
45          vm.prank(user1);
46          systemConfig.updateMarket(
47              "Backpack",
48              address(mockPointToken),
49              0.01 * 1e18,
50              block.timestamp - 1,
51              3600
52          );
53          // Maker settles offer
54          vm.prank(testSeller);
55          deliveryPlace.settleAskMaker(offerAddr, 1000);
56
57          // We are expecting an added token balance of 1000 points token
                 to the taker which they just bought
58          vm.expectEmit(true, true, true, false);
```

```
59          emit AddTokenBalance(testBuyer, address(mockPointToken),
                TokenBalanceType.PointToken, 1000);
60
61          // Taker closes bid
62          vm.startPrank(testBuyer);
63          address stockAddress = GenerateAddress.generateStockAddress(1);
64          deliveryPlace.closeBidTaker(stockAddress);
65          vm.stopPrank();
66
67          // Maker withdraws their revenue from selling 1000 points
68          vm.prank(testSeller);
69          tokenManager.withdraw(
70              address(mockUSDCToken),
71              TokenBalanceType.SalesRevenue
72          );
73
74          // Taker withdraws the points they bought
75          vm.prank(testBuyer);
76          tokenManager.withdraw(
77              address(mockPointToken),
78              TokenBalanceType.PointToken
79          );
80      }
```

Impact

- The taker is unable to withdraw the point tokens they bought, effectively locking their funds in the contract.
- Probably even more serious: The taker gets a wrong token balance for the collateral token which they can withdraw
  - This disrupts the collateral balance of the system and may prevent others to withdraw their collateral

Impact: High Likelihood: High

-> Severity: High

Tools Used

- Manual code review
- Forge unit tests

Recommendations The `MarketPlaceInfo.tokenAddress` contains the address of the point token. Use it instead of the `makerInfo.tokenAddress` in the `DeliveryPlace::closeBidTaker` function:

```
    function closeBidTaker() {
        // ... (existing code)

        (
            OfferInfo memory preOfferInfo,
            MakerInfo memory makerInfo,
+            MarketPlaceInfo memory marketPlaceInfo,

        ) = getOfferInfo(stockInfo.preOffer);
        // ... (other existing code)
        tokenManager.addTokenBalance(
            TokenBalanceType.PointToken,
            _msgSender(),
+           marketPlaceInfo.tokenAddress,
-           makerInfo.tokenAddress,
            pointTokenAmount
        );
    // ... (remaining code)
    }
```