Tadle - Findings Report

Table of contents

Contest Summary

Sponsor: Tadle

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

See more contest details here

Results Summary

Number of findings:

- High: 9
- Medium: 0
- Low: 1

High Risk Findings

H-01. Taker of bid offer will loss assets without any benefit if he calls the DeliveryPlace::settleAskMaker() for partial settlement.

Summary

Taker of bid offer will loss point token & `collateralFee` without any benefit if he calls the `DeliveryPlace::settleAskMaker()` for partial settlement.

Vulnerability Details

Nothing stops a taker of a bid offer to do partial settlement by calling `settleAskTaker()`, but partial settlement results loss of `collateralFee` and Point token for the taker. **NOTE**: To execute the PoC given below properly we need to fix 2 issue of this code, I already submitted the report regarding that issue, you can find that issue with this title: *Call to settleAskTaker() will fail every time due to wrong authority check.* In short you need to correct the authority check in `settleAskTaker()` by changing it from `offerInfo.authority` to `stockInfo.authority`, here. And change the token type from `makerInfo.tokenAddress` to `marketPlaceInfo.tokenAddress`, here, I have already submitted the issue, you can find that with this title: *Wrong token is added to userTokenBalanceMap due to incorrect argument*. I hope you fixed that issue, now lets run the PoC in Premarkets.t.sol contract:

```
    function test_noBenefit() public {
        deal(address(mockPointToken), address(user4), 100e18);

        //@audit User creating a Bid offer, to buy 1000 point
        vm.startPrank(user);
        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Bid,
                OfferSettleType.Turbo
            )
        );
        vm.stopPrank();

        //@audit User4 created a stock to sell 500 point to user's Bid
offer
```

```
        vm.startPrank(user4);
        address offerAddr = GenerateAddress.generateOfferAddress(0);
        preMarktes.createTaker(offerAddr, 500);

        address stock1Addr = GenerateAddress.generateStockAddress(1);
        vm.stopPrank();

        //@audit updateMarket() is called to set the timestamp in
'settlementPeriod' i.e tge was done
        // & we are in now settlementPeriod
        vm.startPrank(user1);
        systemConfig.updateMarket(
            "Backpack",
            address(mockPointToken),
            0.01 * 1e18,
            block.timestamp - 1,
            3600
        );
        //@audit updating the marketPlaceStatus to AskSettling
        systemConfig.updateMarketPlaceStatus(
            "Backpack",
            MarketPlaceStatus.AskSettling
        );
        vm.stopPrank();

        //@audit Now the user came & closed the Bid offer
        vm.prank(user);
        deliveryPlace.closeBidOffer(offerAddr);
        vm.startPrank(user4);
        //@audit user4 tried to settle his Ask type stock so that he can
sell points to the user
        mockPointToken.approve(address(tokenManager), 10000 * 10 ** 18);

        uint pointTokenBalancePrevious = mockPointToken.balanceOf(user4);
        uint usdcTokenBalancePrevious = mockUSDCToken.balanceOf(user4);
        console2.log(
            "Point token balance of user4 before settling: ",
            pointTokenBalancePrevious
        );
        console2.log(
            "USDC token balance of user4 before settling: ",
            usdcTokenBalancePrevious
        );
        console2.log(
            "USDC token balance of user before settling: ",
            mockUSDCToken.balanceOf(address(user))
        );
        console2.log(
            "Point token balance of user before settling: ",
            mockPointToken.balanceOf(address(user))
        );
        deliveryPlace.settleAskTaker(stock1Addr, 300);
        vm.stopPrank();
        uint ownerMakerRefund = tokenManager.userTokenBalanceMap(
```

```
            address(user),
            address(mockUSDCToken),
            TokenBalanceType.MakerRefund
        );
        uint totalUSDCTokenForUser = ownerMakerRefund +
            mockUSDCToken.balanceOf(address(user));
        uint ownerPointToken = tokenManager.userTokenBalanceMap(
            address(user),
            address(mockPointToken),
            TokenBalanceType.PointToken
        );
        uint totalPointTokenForUser = ownerPointToken +
            mockPointToken.balanceOf(address(user));
        console2.log(
            "USDC token balance of user after settling: ",
            totalUSDCTokenForUser
        );
        console2.log(
            "Point token balance of user after settling: ",
            totalPointTokenForUser
        );
        console2.log(
            "USDC token balance of user4 after settling: ",
            mockUSDCToken.balanceOf(address(user4))
        );
        console2.log(
            "Point token balance of user4 after settling: ",
            mockPointToken.balanceOf(address(user4))
        );
    }
```

Logs:

```
Point token balance of user4 before settling:  10000000000000000000
USDC token balance of user4 before settling:  999999999938250000000000000
USDC token balance of user before settling:  999999999900000000000000000
Point token balance of user before settling:  100000000000000000000000000
USDC token balance of user after settling:  100000000001000000000000000
Point token balance of user after settling:  100000003000000000000000000
USDC token balance of user4 after settling:  999999999938250000000000000
Point token balance of user4 after settling:  97000000000000000000
```

Here you can see as the user4 called the `settleAskTaker()` for partial settlement the Point was deducted from his balance, because before settlement his point token balance was: 10000000000000000000 but after settlement his point token balance came to: 97000000000000000000. But for this partial settlement he should have got USDC according to his settlement amount but he did not get anything, before settlement his USDC token balance was: 999999999938250000000000000 & after settlement his USDC token

balance: 999999999938250000000000000 which is same. But if you notice the offer owner Point token balance and USDC token balance, both increased.

Impact

The taker of a bid offer will loss his point token and `collateralFee` if he calls the `settleAskMaker()` for partial settlement.

Tool Used

Manual review, Foundry

Recommendation

It could be design decission to not allow any taker for partial settlement, but if so then the protocol should revert the call immediately if the settlement is partial, so that the taker do not loss his tokens.

Related Links

https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/DeliveryPlace.sol#L335

H-02. TokenManager::withdraw() will revert every time for non-wrappedNativeTokens due to insufficient allowance

Summary

`TokenManager::withdraw()` will revert every time if `_tokenAddress` is not wrappedNativeToken.

Vulnerability Details

`TokenManager::withdraw()` will never work for ERC20 tokens like USDC i.e which is not wrappedNativeToken. Because, when `_tokenAddress` is not wrappedNativeToken then `Rescuable::_safe_transfer_from()` is used to send asset from capitalPool, behind the scene the `_safe_transfer_from()` uses `transferFrom()` to transfer assets which needs approval. But the CapitalPool contract is not approving the TokenManager contract to transfer funds from itself, as a result all call to `withdraw()` for ERC20 tokens will revert.

POC

Run this test in PreMarkets.t.sol:

```
function test_withdraw() public {
        deal(address(mockUSDCToken), address(preMarktes), 1000 * 1e18);
        assertEq(mockUSDCToken.balanceOf(address(preMarktes)), 1000 *
1e18);
        vm.startPrank(user);
        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
```

```
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Ask,
                OfferSettleType.Turbo
            )
        );
        address offerAddr = GenerateAddress.generateOfferAddress(0);
        address stockAddr = GenerateAddress.generateStockAddress(0);
        preMarktes.closeOffer(stockAddr,offerAddr);
        uint userRefundAmount =
tokenManager.userTokenBalanceMap(address(user),address(mockUSDCToken),
TokenBalanceType.MakerRefund);
        console2.log("refund amount: ", userRefundAmount);
        uint balanceBeforeWithdraw = mockUSDCToken.balanceOf(user);
        console2.log("USDC balance before withdraw: ",
balanceBeforeWithdraw);

        //@audit the call to withdraw() will revert
        vm.expectRevert();

tokenManager.withdraw(address(mockUSDCToken),TokenBalanceType.MakerRefund);
```

Logs:

```
├─ [8862] UpgradeableProxy::withdraw(MockERC20Token:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], 4)
    │   ├─ [8343] TokenManager::withdraw(MockERC20Token:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], 4) [delegatecall]
    │   │   ├─ [534] TadleFactory::relatedContracts(4) [staticcall]
    │   │   │   └─ ← [Return] UpgradeableProxy:
[0x76006C4471fb6aDd17728e9c9c8B67d5AF06cDA0]
    │   │   ├─ [2963] MockERC20Token::transferFrom(UpgradeableProxy:
[0x76006C4471fb6aDd17728e9c9c8B67d5AF06cDA0],
0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf, 12000000000000000 [1.2e16])
    │   │   │   └─ ← [Revert]
ERC20InsufficientAllowance(0x6891e60906DEBeA401F670D74d01D117a3bEAD39, 0,
12000000000000000 [1.2e16])
    │   │   └─ ← [Revert] TransferFailed()
    │   └─ ← [Revert] TransferFailed()
```

You can see that the transfer was failed due to insufficient allowance.

Tool Used

Manual review

Recommendation

Call the `CapitalPool::approve()` by passing the `_tokenAddress` before calling `_safe_transfer_from()`.

```
    function withdraw(
        address _tokenAddress,
        TokenBalanceType _tokenBalanceType
    ) external whenNotPaused {
        uint256 claimAbleAmount = userTokenBalanceMap[_msgSender()][
            _tokenAddress
        ][_tokenBalanceType];


        if (claimAbleAmount == 0) {
            return;
        }


        address capitalPoolAddr = tadleFactory.relatedContracts(
            RelatedContractLibraries.CAPITAL_POOL
        );


        if (_tokenAddress == wrappedNativeToken) {

            _transfer(
                wrappedNativeToken,
                capitalPoolAddr,
                address(this),
                claimAbleAmount,
                capitalPoolAddr
            );



IWrappedNativeToken(wrappedNativeToken).withdraw(claimAbleAmount);
            payable(msg.sender).transfer(claimAbleAmount);
        } else {

+           ICapitalPool(capitalPoolAddr).approve(_tokenAddress);
            _safe_transfer_from(
                _tokenAddress,
                capitalPoolAddr,
                _msgSender(),
                claimAbleAmount
            );
        }


        emit Withdraw(
            _msgSender(),
            _tokenAddress,
            _tokenBalanceType,
```

```
                claimAbleAmount
            );
        }
```

Run the mentioned test in POC after adding the approval line, the tokens will be successfully sent.

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/TokenManager.sol#L170-L180

H-03. Wrong token is sent to taker when he closing his bid type stock

Summary

Wrong token is sent to taker when he closing his bid type stock

Vulnerability Details

To close a bid type stock taker calls `closeBidTaker()` in DeliveryPlace.sol contract, the `pointTokenAmount` is sent to taker in this way:

```
tokenManager.addTokenBalance(TokenBalanceType.PointToken, _msgSender(),
makerInfo.tokenAddress, pointTokenAmount);
```

But you can see here that the token was used for transfer is `makerInfo.tokenAddress` which is the token which was used to deposit collateral. The correct token will be `MarketPlaceInfo.tokenAddress`.

Tools Used

Manual review.

Recommendations

Implement this:

```
+    ISystemConfig sytemConfig = tadleFactory.getSystemConfig();
+    MarketPlaceInfo memory marketPlaceInfo =
systemConfig.getMarketPlaceInfo(makerInfo.marketPlace);
-    tokenManager.addTokenBalance(TokenBalanceType.PointToken, _msgSender(),
makerInfo.tokenAddress, pointTokenAmount);
+ tokenManager.addTokenBalance(TokenBalanceType.PointToken, _msgSender(),
marketPlaceInfo.tokenAddress, pointTokenAmount);
```

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/interfaces/ISystemConfig.sol#L142
2. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/DeliveryPlace.sol#L198

H-04. While listing an sub-offer abortOfferStatus is set into memory instead of storage.

Summary

While listing an sub-offer `abortOfferStatus` is set into memory instead of storage.

Vulnerability Details

When a taker wants to list his sub-offer using his stock then - His stock's parent offer's `offerInfo` is fetched.

```
OfferInfo storage offerInfo = offerInfoMap[stockInfo.preOffer];
```

We can see that the location of the struct is storage. After this line the `makerInfo` struct is fetched using the `offerInfo` struct. If the mode of trade is Turbo then `abortOfferStatus` of the original offer is changed to `subOfferListed`. The related code of these actions are here:

```
        if (makerInfo.offerSettleType == OfferSettleType.Turbo) {
            address originOffer = makerInfo.originOffer;
@>          OfferInfo memory originOfferInfo = offerInfoMap[originOffer];


            if (_collateralRate != originOfferInfo.collateralRate) {
                revert InvalidCollateralRate();
            }
            originOfferInfo.abortOfferStatus =
AbortOfferStatus.SubOfferListed;
        }
```

Here, we can see that original offer is fetched from `makerInfo.originOffer`, but unfortunately the `originOfferInfo` is initialized locally instead of globally, for that reason the status of `originOfferInfo.abortOfferStatus` will not not be changed in storage.

Tools Used

Manual review.

Recommendation

```
        if (makerInfo.offerSettleType == OfferSettleType.Turbo) {
            address originOffer = makerInfo.originOffer;
-           OfferInfo memory originOfferInfo = offerInfoMap[originOffer];

+         OfferInfo storage originOfferInfo = offerInfoMap[originOffer];

            if (_collateralRate != originOfferInfo.collateralRate) {
                revert InvalidCollateralRate();
            }
            originOfferInfo.abortOfferStatus =
AbortOfferStatus.SubOfferListed;
        }
```

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/PreMarkets.sol#L335-L343

H-05. Call to settleAskTaker() will fail every time due to wrong authority check.

Summary

Call to `settleAskTaker()` will fail every time due to wrong authority check.

Vulnerability Details

The `DeliveryPlace::settleAskTaker()` is suppoesed to be called by takers of bid offer to release pointToken, amount of point token is caculated by multiplying the `marketPlaceInfo.tokenPerPoint` with the amount of point the taker wants to settle. But taker's call to this function will revert every time because of incorrect authority check, the check is done in the function like this:

```
if (_msgSender() != offerInfo.authority) {
        revert Errors.Unauthorized();
}
```

Here the function expecting the offer's owner as caller, but offer's owner will never call this function, this function is supposed to call by the stock owner i.e taker. Only then point tokens will be deducted from the taker.

POC

Run this test in PreMarkets.t.sol contract, just add an address as user4 and give him enough tokens.

```
        function test_settlingRevert() public {

        //@audit User creating a Bid offer, to buy 1000 point
```

```
        vm.startPrank(user);
        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Bid,
                OfferSettleType.Turbo
            )
        );
        vm.stopPrank();

        //@audit User4 created a stock to sell 500 point to user's Bid
offer
        vm.startPrank(user4);
        address offerAddr = GenerateAddress.generateOfferAddress(0);
        preMarktes.createTaker(offerAddr, 500);

        address stock1Addr = GenerateAddress.generateStockAddress(1);
        vm.stopPrank();

        //@audit updateMarket() is called to set the timestamp in
'settlementPeriod' i.e tge was done
        // & we are in now settlementPeriod
        vm.startPrank(user1);
        systemConfig.updateMarket(
            "Backpack",
            address(mockPointToken),
            0.01 * 1e18,
            block.timestamp - 1,
            3600
        );
        //@audit updating the marketPlaceStatus to AskSettling
        systemConfig.updateMarketPlaceStatus("Backpack",
MarketPlaceStatus.AskSettling);
        vm.stopPrank();

        //@audit Now the user came & closed the Bid offer
        vm.prank(user);
        deliveryPlace.closeBidOffer(offerAddr);
        vm.startPrank(user4);
        //@audit user4 tried to settle his Ask type stock so that he can
sell points to the user
        mockPointToken.approve(address(tokenManager), 10000 * 10 ** 18);
        //@audit But it is reverting
        vm.expectRevert(Errors.Unauthorized.selector);
        deliveryPlace.settleAskTaker(stock1Addr, 500);
        vm.stopPrank();
    }
```

Tool Used

Manual review, Foundry

Recommendation

```
-   if (_msgSender() != offerInfo.authority) {
        revert Errors.Unauthorized();
    }

+   if (_msgSender() != stockInfo.authority) {
        revert Errors.Unauthorized();
    }
```

Related Links

H-06. Incorrect logic in abortBidTaker() will return wrong depositAmount

Summary

Incorrect logic will return wrong `depositAmount` in `PreMarkets::abortBidTaker()`.

Vulnerability Details

In `PreMarkets::abortBidTaker()` the deposit amount is calculated incorrectly. The current logic looks like this:

```
        uint256 depositAmount = stockInfo.points.mulDiv(
            preOfferInfo.points,
            preOfferInfo.amount,
            Math.Rounding.Floor
        );
```

Assume, stockInfo.points = 100, preOfferInfo.points = 1000 & preOfferInfo.amount = 2000. It means price of 1 point is 2. If we follow the current implementation & put it to Math.sol::mulDiv() then we will get this:

```
➜ mulDiv(100,1000,2000)
Type: uint256
├ Hex: 0x32
├ Hex (full word): 0x32
└ Decimal: 50
```

But it is wrong, for 100 points the deposit amount should be 200 because price of 1 point is 2.

Impact

Wrong `depositAmount` will be calculated.

Tools Used

Manual review, Chisel

Recommendations

```
        uint256 depositAmount = stockInfo.points.mulDiv(
-           preOfferInfo.points,
-            preOfferInfo.amount,
            Math.Rounding.Floor
        );
        uint256 depositAmount = stockInfo.points.mulDiv(
+            preOfferInfo.amount,
+            preOfferInfo.points,
            Math.Rounding.Floor
        );
```

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/PreMarkets.sol#L671-L675

H-07. Incorrect approval in TokenManager::_transfer() will result in revert.

Summary

Incorrect approval in `TokenManager::_transfer()` will result in revert when `_from` is capitalPool address & allowance for tokenManager for that token is 0

Vulnerability Details

As of the logic of `TokenManager::_transfer()` if `_from` is capitalPool address & allowance for tokenManager for that token is 0 then the function will execute this code:

```
ICapitalPool(_capitalPoolAddr).approve(address(this));
```

Notice that the argument of `approve()` is `address(this)` i.e tokenManager contract, but if you the `CapitalPool::approve()` the function takes a token address as argument, not tokenManager's address.

```
    /**
     * @dev Approve token for token manager
```

```
     * @notice only can be called by token manager
     * @param tokenAddr address of token
     */
    function approve(address tokenAddr) external {
            address tokenManager = tadleFactory.relatedContracts(
            RelatedContractLibraries.TOKEN_MANAGER
        );
        (bool success, ) = tokenAddr.call(
            abi.encodeWithSelector(
                APPROVE_SELECTOR,
                tokenManager,
                type(uint256).max
            )
        );


        if (!success) {
            revert ApproveFailed();
        }
    }
```

As a result all call to `_transfer()` will fail because `APPROVE_SELECTOR` is not a selector of `tokenManager` contract.

Impact

All call to `_transfer()` will fail.

Tools Used

Manual review.

Recommendations

```
        if (
            _from == _capitalPoolAddr &&
            IERC20(_token).allowance(_from, address(this)) == 0x0
        ) {
-            ICapitalPool(_capitalPoolAddr).approve(address(this));
+            ICapitalPool(_capitalPoolAddr).approve(_token);
        }
```

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/TokenManager.sol#L243-L248

H-08. User can drain the CapitalPool contract.

Summary

Due to not deducting the claimed amount from userTokenBalanceMap mapping after withdrawing an user can drain the vault by calling the `TokenManager::withdraw()` multiple times.

Vulnerability Details

In `TokenManager::withdraw()` the claimable amount is fetched using a mapping of TokenManagerStorage contract.

```
/// @dev user token balance can be claimed by user.
    /// @dev userTokenBalanceMap[accountAddress][tokenAddress]
[tokenBalanceType]
    mapping(address => mapping(address => mapping(TokenBalanceType =>
uint256)))
        public userTokenBalanceMap;
```

However after transferring the token the mapping is not updated i.e the transferred amount is not deducted from that mapping, as a result any one can call the `withdraw()` multiple time and drain all the funds. To show this in POC just assume CapitalPool contract has a balance of 0.036000000000000000 USDC, keeping the balance that low so that I can hust prove the bug. Run this test in PreMarkets.t.sol:

```
    function test_withdraw() public {
        deal(address(mockUSDCToken), address(capitalPool),
36000000000000000);
        assertEq(mockUSDCToken.balanceOf(address(capitalPool)),
36000000000000000);
        vm.startPrank(user);
        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Ask,
                OfferSettleType.Turbo
            )
        );
        address offerAddr = GenerateAddress.generateOfferAddress(0);
        address stockAddr = GenerateAddress.generateStockAddress(0);
        preMarktes.closeOffer(stockAddr, offerAddr);
        uint userRefundAmount = tokenManager.userTokenBalanceMap(
            address(user),
            address(mockUSDCToken),
            TokenBalanceType.MakerRefund
        );
        console2.log("refund amount: ", userRefundAmount);
```

```
        uint balanceBeforeWithdraw = mockUSDCToken.balanceOf(user);
        console2.log("USDC balance before withdraw: ",
balanceBeforeWithdraw);
        console2.log(
            "capitalPool balance before withdraw: ",
mockUSDCToken.balanceOf(address(capitalPool))
        );
        for(uint i; i < 4; i++){
            tokenManager.withdraw(
            address(mockUSDCToken),
            TokenBalanceType.MakerRefund
        );
        }

        console2.log(
            "capitalPool balance at the end: ",
mockUSDCToken.balanceOf(address(capitalPool))
        );
        console2.log("USDC balance after withdraw: ",
balanceBeforeWithdraw);
    }
```

Logs:

```
  refund amount:  12000000000000000
  USDC balance before withdraw:  999999999880000000000000000
  capitalPool balance before withdraw:  48000000000000000
  capitalPool balance at the end:  0
  USDC balance after withdraw:  999999999880000000000000000
```

Impact

User can drain the CapitalPool contract by calling withdraw() multiple times.

Tool Used

Manual review, Foundry

Recommendation

Deduct the claimed amount from the `userTokenBalanceMap` after transferring tokens.

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/TokenManager.sol#L137-L189

H-09. Wrong token is added to userTokenBalanceMap due to incorrect argument

Summary

Wrong token is added to `userTokenBalanceMap` due to incorrect argument in `tokenManager::addTokenBalance()` call.

Vulnerability Details

The `DeliveryPlace::settleAskTaker()` is supposed to called by a Ask type stock owner to settle points for a bid offer [that offer for which this stock was created]. This `settleAskTaker()` is responsible to take pointTokens from the stock owner and add that in `userTokenBalanceMap` for TokenBalanceType.PointToken. However while adding the balance in the mapping USDC token was used instead of Point token, as a result USDC token will be added in the mapping instead of Point token. **NOTE**: To execute the PoC given below we need to fix a issue of this code, I already submitted the report regarding that issue, you can find that issue with this title: *Call to settleAskTaker() will fail every time due to wrong authority check*. In short you need to correct the authority check in `settleAskTaker()` by changing it from `offerInfo.authority` to `stockInfo.authority`, here.I hope you fixed that issue, now lets run the PoC in Premarkets.t.sol contract:

```
    function test_wrongToken() public {
        deal(address(mockPointToken), address(user4), 100e18);

        //@audit User creating a Bid offer, to buy 1000 point
        vm.startPrank(user);
        preMarktes.createOffer(
            CreateOfferParams(
                marketPlace,
                address(mockUSDCToken),
                1000,
                0.01 * 1e18,
                12000,
                300,
                OfferType.Bid,
                OfferSettleType.Turbo
            )
        );
        vm.stopPrank();

        //@audit User4 created a stock to sell 500 point to user's Bid
offer
        vm.startPrank(user4);
        address offerAddr = GenerateAddress.generateOfferAddress(0);
        preMarktes.createTaker(offerAddr, 500);

        address stock1Addr = GenerateAddress.generateStockAddress(1);
        vm.stopPrank();

        //@audit updateMarket() is called to set the timestamp in
```

```
'settlementPeriod' i.e tge was done
        // & we are in now settlementPeriod
        vm.startPrank(user1);
        systemConfig.updateMarket(
            "Backpack",
            address(mockPointToken),
            0.01 * 1e18,
            block.timestamp - 1,
            3600
        );
        //@audit updating the marketPlaceStatus to AskSettling
        systemConfig.updateMarketPlaceStatus(
            "Backpack",
            MarketPlaceStatus.AskSettling
        );
        vm.stopPrank();

        //@audit Now the user came & closed the Bid offer
        vm.prank(user);
        deliveryPlace.closeBidOffer(offerAddr);
        vm.startPrank(user4);
        //@audit user4 tried to settle his Ask type stock so that he can
sell points to the user
        mockPointToken.approve(address(tokenManager), 10000 * 10 ** 18);

        console2.log(
            "USDC token balance of user before settling: ",
            mockUSDCToken.balanceOf(address(user))
        );
        console2.log(
            "Point token balance of user before settling: ",
            mockPointToken.balanceOf(address(user))
        );
        deliveryPlace.settleAskTaker(stock1Addr, 300);
        vm.stopPrank();
        uint ownerMakerRefund = tokenManager.userTokenBalanceMap(
            address(user),
            address(mockUSDCToken),
            TokenBalanceType.MakerRefund
        );
        uint totalUSDCTokenForUser = ownerMakerRefund +
            mockUSDCToken.balanceOf(address(user));
        uint ownerPointToken = tokenManager.userTokenBalanceMap(
            address(user),
            address(mockPointToken),
            TokenBalanceType.PointToken
        );
        uint totalPointTokenForUser = ownerPointToken +
            mockPointToken.balanceOf(address(user));
        console2.log(
            "USDC token balance of user after settling: ",
            totalUSDCTokenForUser
        );
        console2.log(
```

```
            "Point token balance of user after settling: ",
            totalPointTokenForUser
        );
    }
```

Logs:

```
  USDC token balance of user before settling:  999999999900000000000000000
  Point token balance of user before settling:  100000000000000000000000000
  USDC token balance of user after settling:  100000000001000000000000000
  Point token balance of user after settling:  100000000000000000000000000
```

As you can USDC balance was increased instead of Point token.

Impact

Wrong token will added to `userTokenBalanceMap` mapping.

Tool Used

Manual review, Foundry

Recommendation

```
            tokenManager.addTokenBalance(
                TokenBalanceType.PointToken,
                offerInfo.authority,
-               makerInfo.tokenAddress,
+               marketPlaceInfo.tokenAddress,
                settledPointTokenAmount
            );
```

This is the log of same test after editing the code as above:

```
 USDC token balance of user before settling:  999999999900000000000000000
  Point token balance of user before settling:  100000000000000000000000000
  USDC token balance of user after settling:  100000000001000000000000000
  Point token balance of user after settling:  100000003000000000000000000
```

You can see the Point token balance was increased.

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/DeliveryPlace.sol#L384-L389

Low Risk Findings

L-01. referrerRate is set incorrectly in updateRefferInfo()

Summary

`referralInfo.referrerRate` is set incorrectly in `SystemConfig::updateRefferInfo()`

Vulnerability Details

As per documentation referrerRate can be up to a maximum of 30%, but here it is setting to minimum 30% i.e totally opposite & the `baseReferralRate` is set to 30% by default, which means referrerRate can be as high as possible.

```
if (_referrerRate < baseReferralRate) {
            revert InvalidReferrerRate(_referrerRate);
}
```

Tool Used

Manual review

Recommendation

```
-   if (_referrerRate < baseReferralRate) {
             revert InvalidReferrerRate(_referrerRate);
    }

+   if (_referrerRate > baseReferralRate || _referrerRate == 0 ) {
            revert InvalidReferrerRate(_referrerRate);
}
```

I am assuming `_referrerRate` can't be 0 atleast.

Related Links

1. https://github.com/Cyfrin/2024-08-tadle/blob/04fd8634701697184a3f3a5558b41c109866e5f8/src/core/SystemConfig.sol#L54