PoolTogether TwabRewards contest Findings & Analysis Report

2022-02-03

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of PoolTogether TwabRewards contest smart contract system written in Solidity. The code contest took place between December 9—December 12 2021.

Wardens

28 Wardens contributed reports to the PoolTogether TwabRewards contest:

1. WatchPug (jtp and ming)
2. leastwood
3. harleythedog
4. defsec
5. kenzo

6. certora
7. sirhashalot
8. csanuragjain
9. 0x0x0x
10. pauliax
11. gpersoon
12. GiveMeTestEther
13. hubble (ksk2345 and shri4net)
14. pmerkleplant
15. gzeon
16. cmichel
17. johnnycash
18. Meta0xNull
19. 0xabc
20. kemmio
21. 0x421f
22. robee
23. Jujic
24. yeOlde
25. danb

This contest was judged by LSDan (ElasticDAO).

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 21 unique vulnerabilities and 38 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 7 received a risk rating in the category of HIGH severity, 5 received a risk rating in the category of MEDIUM severity, and 9 received a risk rating in the category of LOW severity.

C4 analysis also identified 2 non-critical recommendations and 15 gas optimizations.

Scope

The code under review can be found within the C4 PoolTogether TwabRewards contest repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 499 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (7)

[H-01] `createPromotion()` Lack of input validation for `_epochDuration` can potentially freeze promotion creator's funds

Submitted by WatchPug

https://github.com/pooltogether/v4-periphery/blob/0e94c54774a6fce29daf9cb23353208f80de63eb/contracts/Twa bRewards.sol#L88-L116

```
    function createPromotion(
        address _ticket,
        IERC20 _token,
        uint216 _tokensPerEpoch,
        uint32 _startTimestamp,
        uint32 _epochDuration,
        uint8 _numberOfEpochs
    ) external override returns (uint256) {
        _requireTicket(_ticket);

        uint256 _nextPromotionId = _latestPromotionId + 1;
        _latestPromotionId = _nextPromotionId;

        _promotions[_nextPromotionId] = Promotion(
            msg.sender,
            _ticket,
            _token,
            _tokensPerEpoch,
            _startTimestamp,
            _epochDuration,
            _numberOfEpochs
        );

        _token.safeTransferFrom(msg.sender, address(this), _tokensPe

        emit PromotionCreated(_nextPromotionId);

        return _nextPromotionId;
    }
```

In the current implementation of `createPromotion()`, `_epochDuration` is allowed to be `0`.

However, when `_epochDuration = 0`, it will be impossible for users to claim the rewards, and the promotion creator won't be able to cancel it.

Proof of Concept

1. Alice called `createPromotion()` to create a promotion with the following parameters:

    - _token: `USDC`
    - _tokensPerEpoch: `10,000`
    - _epochDuration: `0`
    - _numberOfEpochs: `10`
2. `100,000 USDC` was transferred from Alice to the `TwabRewards` contract;
3. Users tries to `claimRewards()` but the transaction always revert at `_ticket.getAverageTotalSuppliesBetween()` -> `TwabLib.getAverageBalanceBetween()` due to div by 0.
4. Alice tries to `cancelPromotion()` to retrieve the funds, but it always reverts at `_requirePromotionActive()` since the promotion already ended.

As a result, Alice's `100,000 USDC` is frozen in the contract.

Recommendation

Consider adding `require(_epochDuration > 0)` in `createPromotion()`.

[PierrickGT (PoolTogether) marked as duplicate](#):

> Duplicate of https://github.com/code-423n4/2021-12-pooltogether-findings/issues/29

[LSDan (judge) commented](#):

> I do not consider this to be a duplicate of #29 because the warden in #29 does not mention this specific failure case. This is indeed an easy to encounter bug that can be triggered as the result of a user error or a frontend bug. Loss of all funds for the promotion would be the result.

PierrickGT (PoolTogether) confirmed and resolved:

> Implemented the suggested require: https://github.com/pooltogether/v4-periphery/blob/e0010b689fb170daac77af5f62abba7ca1397524/contracts/TwabRewards.sol#L126

[H-02] Backdated _startTimestamp can lead to loss of funds

Submitted by csanuragjain, also found by defsec, leastwood, and pauliax

Impact

This can lead to loss of funds as there is no recovery function of funds stuck like this

Proof of Concept

1. User A creates a new promotion using createPromotion function. By mistake he provides 1 year ago value for `\_startTimestamp` with promotion duration as 6 months
2. Since there is no check to see that `\_startTimestamp > block.timestamp` so this promotion gets created
3. User cannot claim this promotion if they were not having promotion tokens in the 1 year old promotion period. This means promotion amount remains with contract
4. Even promotion creator cannot claim back his tokens since promotion end date has already passed so `cancelPromotion` will fail
5. As there is no recovery token function in contract so even contract cant transfer this token and the tokens will remain in this contract with no one able to claim those

Recommended Mitigation Steps

Add below check in the `createPromotion` function

```
function createPromotion(
    address _ticket,
    IERC20 _token,
    uint216 _tokensPerEpoch,
    uint32 _startTimestamp,
    uint32 _epochDuration,
    uint8 _numberOfEpochs
) external override returns (uint256) {
    require(_startTimestamp>block.timestamp,"should be after cur
}
```

PierrickGT (PoolTogether) confirmed and disagreed with severity:

> It would indeed be an unfortunate event and we will implement this require. That being said, funds of the promotion creator would be at risk, because of an error he made, but not funds of a user, so I consider this bug as being of severity 2 (Med Risk) and not 3 (High Risk).

LSDan (judge) commented:

> Per the Judge Onboarding document provided by Code423n4, this qualifies as a high risk issue. A UI bug or simple mistake could cause complete loss of funds as sponsor acknowledged.

> `3 — High (H): vulns have a risk of 3 and are considered "High" severity when assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).`

[H-03] Continue claiming reqrds after numberOfEpochs are over

Submitted by gpersoon, also found by 0xabc, csanuragjain, harleythedog, kenzo, and leastwood

Impact

When claiming rewards via `claimRewards()`, the function `\_calculateRewardAmount()` is called. The function `\_calculateRewardAmount()` has a check to make sure the epoch is over

```
    require(block.timestamp > _epochEndTimestamp, "TwabRewards/epc
```

However neither functions check if the `\_epochId` is within the range of the reward epochs. Ergo it is possible to continue claiming rewards after the reward period is over. This only works as long as there are enough tokens in the contract. But this is the case when not everyone has claimed, or other rewards use the same token.

The proof of concept contains a simplified version of the contract, and shows how this can be done. When run in remix you get the following output, while there is only 1 epoch. `console.log: Claiming for epoch 1 1 Claiming for epoch 2 1 Claiming for epoch 3 1 Claiming for epoch 4 1 Claiming for epoch 5 1`

Proof of Concept

```
 // SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.6;
import "hardhat/console.sol";

contract TwabRewards {

    struct Promotion {
        uint216 tokensPerEpoch;
        uint32 startTimestamp;
        uint32 epochDuration;
        uint8 numberOfEpochs;
    }
    mapping(uint256 => Promotion) internal _promotions;
    uint256 internal _latestPromotionId;
    mapping(uint256 => mapping(address => uint256)) internal _cl

    constructor() {
        uint id=createPromotion(1,uint32(block.timestamp)-10,1,1
        claimRewards(id,1);
        claimRewards(id,2);
        claimRewards(id,3);
        claimRewards(id,4);
        claimRewards(id,5);
    }

    function createPromotion(uint216 _tokensPerEpoch,uint32 _sta
        uint256 _nextPromotionId = _latestPromotionId + 1;
        _latestPromotionId = _nextPromotionId;
        _promotions[_nextPromotionId] = Promotion(_tokensPerEpoc
        return _nextPromotionId;
    }

    function claimRewards(
        uint256 _promotionId,
        uint256 _epochId
    ) public  returns (uint256) {
        Promotion memory _promotion = _getPromotion(_promotionIc
        address _user=address(0);
        uint256 _rewardsAmount;
        uint256 _userClaimedEpochs = _claimedEpochs[_promotionIc

        for (uint256 index = 0; index < 1; index++) {
            require(
                !_isClaimedEpoch(_userClaimedEpochs, _epochId),
                "TwabRewards/rewards-already-claimed"
            );
            _rewardsAmount += _calculateRewardAmount(_promotion,
            _userClaimedEpochs = _updateClaimedEpoch(_userClaime
        }
        _claimedEpochs[_promotionId][_user] = _userClaimedEpochs
        console.log("Claiming for epoch",_epochId,_rewardsAmount
        return _rewardsAmount;
    }

    function getPromotion(uint256 _promotionId) public view  ret
        return _getPromotion(_promotionId);
    }
  function _getPromotion(uint256 _promotionId) internal view ret
        return _promotions[_promotionId];
    }

    function _isClaimedEpoch(uint256 _userClaimedEpochs, uint256
    {
        return (_userClaimedEpochs >> _epochId) & uint256(1) ==
    }

function _calculateRewardAmount(
        Promotion memory _promotion,
        uint256 _epochId
    ) internal view returns (uint256) {
        uint256 _epochDuration = _promotion.epochDuration;
        uint256 _epochStartTimestamp = _promotion.startTimestamp
        uint256 _epochEndTimestamp = _epochStartTimestamp + _epc
        require(block.timestamp > _epochEndTimestamp, "TwabRewar
        return 1;
    }

function _updateClaimedEpoch(uint256 _userClaimedEpochs, uint25
        return _userClaimedEpochs | (uint256(1) << _epochId);
    }

    function _getCurrentEpochId(Promotion memory _promotion) int
```

```
        return (block.timestamp - _promotion.startTimestamp) / _
    }

    function _getRemainingRewards(Promotion memory _promotion) i
        // _tokensPerEpoch * _numberOfEpochsLeft
        return
            _promotion.tokensPerEpoch *
            (_promotion.numberOfEpochs - _getCurrentEpochId(_prc
    }

}
```

Recommended Mitigation Steps

In the function `\_calculateRewardAmount()` add something like the following in the beginning after the require. `if ( \_epochId >= \_promotion.numberOfEpochs) return 0;`

[PierrickGT (PoolTogether) confirmed]

[H-04] cancelPromotion is too rigorous

Submitted by gpersoon, also found by 0x0x0x, gzeon, harleythedog, hubble, and kenzo

Impact

When you cancel a promotion with `cancelPromotion()` then the promotion is complete deleted. This means no-one can claim any rewards anymore, because `\_promotions\[\_promotionId]` no longer exists.

It also means all the unclaimed tokens (of the previous epochs) will stay locked in the contract.

Proof of Concept

https://github.com/pooltogether/v4-periphery/blob/b520faea26bcf60371012f6cb246aa149abd3c7d/contracts/TwabRewards.sol#L119-L138

```
    function cancelPromotion(uint256 _promotionId, address _to) ...
        ...
        uint256 _remainingRewards = _getRemainingRewards(_promotion)
        delete _promotions[_promotionId];
```

Recommended Mitigation Steps

In the function `cancelPromotion()` lower the `numberOfEpochs` or set a state variable, to allow user to claim their rewards.

[PierrickGT (PoolTogether) confirmed]

[H-05] Malicious tickets can lead to the loss of all tokens

Submitted by johnnycash, also found by WatchPug, csanuragjain, gpersoon, gzeon, harleythedog, kemmio, kenzo, leastwood, and pauliax

Impact

It allows an attacker to retrieve all the tokens of each promotions.

Analysis

Anyone can create a new promotion using `createPromotion()`. An attacker can create a new malicious promotion with the following parameters:

- the address of a malicious ticket smart contract
- the token address from the targeted promotion(s)
- optionally, `_numberOfEpochs` equal to 0 to create this promotion for free

The only verification made on the ticket address given by [_requireTicket()] is that the smart contract must implement the `ITicket` interface.

The attacker can then call `claimRewards()` with its wallet address, the malicious promotion id and a single _epochId for the sake of clarity.

1. `_calculateRewardAmount()` is first called to get the reward amount with the following formula `(_promotion.tokensPerEpoch * _ticket.getAverageBalanceBetween()) / _ticket.getAverageTotalSuppliesBetween()`. The malicious ticket can return an arbitrary `_averageBalance` and an `_averageTotalSupplies` of 1, leading to an arbitrary large reward amount.
2. `_promotion.token.safeTransfer(_user, _rewardsAmount)` is called. It transfers the amount of tokens previously computed to the attacker.

The attacker receives the tokens of other promotions without having spent anything.

Proof of Concept

The malicious smart contract is a copy/paste of TicketHarness.sol and Ticket.sol with the following changes:

```
    /// @inheritdoc ITicket
    function getAverageTotalSuppliesBetween(
        uint64[] calldata _startTimes,
        uint64[] calldata _endTimes
    ) external view override returns (uint256[] memory) {
        uint256[] memory _balances = new uint256[](1);
        _balances[0] = uint256(1);
        return _balances;
    }

    /// @inheritdoc ITicket
    function getAverageBalanceBetween(
        address _user,
        uint64 _startTime,
        uint64 _endTime
    ) external view override returns (uint256) {
        return 1337;
    }
```

The test for HardHat is:

```
    describe('exploit()', async () => {
        it('this shouldnt happen', async () => {
            const promotionIdOne = 1;
            const promotionIdTwo = 2;

            await expect(createPromotion(ticket.address))
                .to.emit(twabRewards, 'PromotionCreated')
                .withArgs(promotionIdOne);

            let evilTicketFactory = await getContractFactory('EvilTi
            let evilTicket = await evilTicketFactory.deploy('EvilTic
            let createPromotionTimestamp = (await ethers.provider.ge
            await expect(twabRewards.connect(wallet2).createPromotic
                evilTicket.address,
                rewardToken.address,
                tokensPerEpoch,
                createPromotionTimestamp,
                1,//epochDuration,
                0,//epochsNumber,
            )).to.emit(twabRewards, 'PromotionCreated')
                .withArgs(promotionIdTwo);

            await increaseTime(100);
            const epochIds = ['100'];
            await twabRewards.connect(wallet2).claimRewards(wallet2.
        });
    });
```

It results in the following error:

```
    1) TwabRewards
        exploit()
            this shouldnt happen:
        Error: VM Exception while processing transaction: reverted w
        at TwabRewardsHarness.verifyCallResult (@openzeppelin/contra
        at TwabRewardsHarness.functionCallWithValue (@openzeppelin/c
        at TwabRewardsHarness.functionCall (@openzeppelin/contracts/
        at TwabRewardsHarness._callOptionalReturn (@openzeppelin/cor
        at TwabRewardsHarness.safeTransfer (@openzeppelin/contracts/
        at TwabRewardsHarness.claimRewards (contracts/TwabRewards.sc
```

Recommended Mitigation Steps

Maybe add a whitelist of trusted tickets?

PierrickGT (PoolTogether) confirmed

[H-06] Rewards can be claimed multiple times

Submitted by johnnycash, also found by certora, cmichel, gpersoon, gzeon, harleythedog, kemmio, kenzo, sirhashalot, and 0x421f

Impact

An attacker can claim its reward 256 * `epochDuration` seconds after the timestamp at which the promotion started. The vulnerability allows him to claim a reward several times to retrieve all the tokens associated to the promotion.

Analysis

`claimRewards()` claim rewards for a given promotion and epoch. In order to prevent a user from claiming a reward multiple times, the mapping _claimedEpochs keeps track of claimed rewards per user:

```
    /// @notice Keeps track of claimed rewards per user.
    /// @dev _claimedEpochs[promotionId][user] => claimedEpochs
    /// @dev We pack epochs claimed by a user into a uint256. So we
    mapping(uint256 => mapping(address => uint256)) internal _claime
```

(The comment is wrong, epochs are packed into a uint256 which allows **256** epochs to be stored).

`_epochIds` is an array of `uint256`. For each `_epochId` in this array, `claimRewards()` checks that the reward associated to this `_epochId` isn't already claimed thanks to `_isClaimedEpoch()`. _isClaimedEpoch() checks that the bit `_epochId` of `_claimedEpochs` is unset:

```
    (_userClaimedEpochs >> _epochId) & uint256(1) == 1;
```

However, if `_epochId` is greater than 255, `_isClaimedEpoch()` always returns false. It allows an attacker to claim a reward several times.

_calculateRewardAmount() just makes use of `_epochId` to tell whether the promotion is over.

Proof of Concept

The following test should result in a reverted transaction, however the transaction succeeds.

```
    it('should fail to claim rewards if one or more epochs have alre
        const promotionId = 1;

        const wallet2Amount = toWei('750');
        const wallet3Amount = toWei('250');

        await ticket.mint(wallet2.address, wallet2Amount);
        await ticket.mint(wallet3.address, wallet3Amount);

        await createPromotion(ticket.address);
        await increaseTime(epochDuration * 257);

        await expect(
            twabRewards.claimRewards(wallet2.address, promotionId, |
        ).to.be.revertedWith('TwabRewards/rewards-already-claimed');
    });
```

Recommended Mitigation Steps

A possible fix could be to change the type of `_epochId` to `uint8` in:

- `_calculateRewardAmount()`
- `_updateClaimedEpoch()`
- `_isClaimedEpoch()`

and change the type of `_epochIds` to `uint8[]` in `claimRewards()`.

PierrickGT (PoolTogether) confirmed

[H-07] Contract does not work with fee-on transfer tokens

Submitted by pmerkleplant, also found by GiveMeTestEther, WatchPug, and defsec

Impact

There exist ERC20 tokens that charge a fee for every transfer.

This kind of token does not work correctly with the `TwabRewards` contract as the rewards calculation for an user is based on `promotion.tokensPerEpoch` (see line [320](320)).

However, the actual amount of tokens the contract holds could be less than `promotion.tokensPerEpoch * promotion.numberOfEpochs` leading to not claimable rewards for users claiming later than others.

Recommended Mitigation Steps

To disable fee-on transfer tokens for the contract, add the following code in `createPromotion` around line 11:

```
    uint256 oldBalance = _token.balanceOf(address(this));
    _token.safeTransferFrom(msg.sender, address(this), _tokensPerEpc
    uint256 newBalance = _token.balanceOf(address(this));
    require(oldBalance + _tokenPerEpoch * _numberOfEpochs == newBala
```

[PierrickGT (PoolTogether) confirmed]()

[LSDan (judge) commented]():

> This issue results in a direct loss of funds and can happen easily.

> `3 — High (H): vulns have a risk of 3 and are considered "High" severity when assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).`

Medium Risk Findings (5)

[M-01] `cancelPromotion()` Unable to cancel unstarted promotions

Submitted by WatchPug, also found by kenzo, and certora

For unstarted promotions, `cancelPromotion()` will revert at `block.timestamp - _promotion.startTimestamp` in `_getCurrentEpochId()`.

Call stack: `cancelPromotion()` -> `_getRemainingRewards()` -> `_getCurrentEpochId()`.

https://github.com/pooltogether/v4-periphery/blob/0e94c54774a6fce29daf9cb23353208f80de63eb/contracts/TwabRewards.sol#L331-L336

```
function _getRemainingRewards(Promotion memory _promotion) inter
    // _tokensPerEpoch * _numberOfEpochsLeft
    return
        _promotion.tokensPerEpoch *
        (_promotion.numberOfEpochs - _getCurrentEpochId(_promoti
}
```

https://github.com/pooltogether/v4-periphery/blob/0e94c54774a6fce29daf9cb23353208f80de63eb/contracts/TwabRewards.sol#L276-L279

```
function _getCurrentEpochId(Promotion memory _promotion) interna
    // elapsedTimestamp / epochDurationTimestamp
    return (block.timestamp - _promotion.startTimestamp) / _prom
}
```

Recommendation

Consider checking if `_promotion.startTimestamp > block.timestamp` and refund `_promotion.tokensPerEpoch * _promotion.numberOfEpochs` in `cancelPromotion()`.

PierrickGT (PoolTogether) confirmed

[M-02] getRewardsAmount doesn’t check epochs haven’t been claimed

Submitted by harleythedog

Impact

In ITwabRewards.sol, it is claimed that `getRewardsAmount` should account for epochs that have already been claimed, and not include these epochs in the total amount (indeed, there is a line that says `@dev Will be 0 if user has already claimed rewards for the epoch.`)

However, no such check is done in the implementation of `getRewardsAmount`. This means that users will be shown rewardAmounts that are higher than they should be, and users will be confused when they are transferred fewer tokens than they are told they will. This would cause confusion, and people may begin to mistrust the contract since they think they are being transferred fewer tokens than they are owed.

Proof of Concept

See the implementation of `getRewardsAmount` here:
https://github.com/pooltogether/v4-periphery/blob/b520faea26bcf60371012f6cb246aa149abd3c7d/contracts/TwabRewards.sol#L209

Notice that there are no checks that the epochs have not already been claimed. Compare this to `claimRewards` which *does* check for epochs that have already been claimed with the following require statement:

```
    require(!_isClaimedEpoch(_userClaimedEpochs, _epochId), "TwabRev
```

A similar check should be added `getRewardsAmount` so that previously claimed epochs are not included in the sum.

Recommended Mitigation Steps

Add a similar check for previously claimed epochs as described above.

[PierrickGT (PoolTogether) confirmed]

[M-03] Dust Token Balances Cannot Be Claimed By An `admin` Account

Submitted by leastwood, also found by 0x0x0x

Impact

Users who have a small claim on rewards for various promotions, may not feasibly be able to claim these rewards as gas costs could outweigh the sum they receive in return. Hence, it is likely that a dust balance accrues overtime for tokens allocated for various promotions. Additionally, the `_calculateRewardAmount` calculation may result in truncated results, leading to further accrual of a dust balance. Therefore, it is useful that these funds do not go to waste.

Proof of Concept

[https://github.com/pooltogether/v4-periphery/blob/b520faea26bcf60371012f6cb246aa149abd3c7d/contracts/TwabRewards.sol#L162-L191]

Recommended Mitigation Steps

Consider allowing an `admin` account to skim a promotion's tokens if it has been inactive for a certain length of time. There are several potential implementations, in varying degrees of complexity. However, the solution should attempt to maximise simplicity while minimising the accrual of dust balances.

[PierrickGT (PoolTogether) confirmed]

[M-04] Unsafe uint64 casting may overflow

Submitted by sirhashalot

Impact

The `\_calculateRewardAmount` function casts epoch timestamps from uint256 to uint64 and these may overflow. The epochStartTimestamp value is a function of the user-supplied `\_epochId` value, which could be extremely large (up to 2**255 — 1). While Solidity 0.8.x checks for overflows on arithmetic operations, it does not do so for casting — the OpenZeppelin SafeCast library offers this. The overflow condition could cause `\_epochStartTimestamp > \_epochEndTimestamp`, which the Ticket.sol getAverageBalanceBetween may not be expected to handle. The `\_epochStartTimestamp` could overflow to have a value before the actual start of the promotion, also impacting the rewards calculation.

Proof of Concept

There are 4 uint64 casting operations in the `\_calculateRewardAmount` function of `TwabRewards.sol`: https://github.com/pooltogether/v4-periphery/blob/b520faea26bcf60371012f6cb246aa149abd3c7d/contracts/TwabRewards.sol#L304-L312

Recommended Mitigation Steps

While requiring `\_epochId <= 255` may help, it does not remove the issue entirely, because a very large `\_epochDuration` value can still cause an overflow in the product `(\_epochDuration \* \_epochId)` used in `\_epochStartTimestamp`. However, other options exist:

1. Making these uint256 variables of type uint64 and therefore removing the casting of uint256 to the small uint64 would remove this risk and probably be the most gas-efficient solution.
2. Add `require(_epochEndTimestamp > _epochStartTimestamp);` to line 299, next to the existing require statement and before the uint64 casting operations
3. Use the OpenZeppelin SafeCast library to prevent unexpected overflows.

[PierrickGT (PoolTogether) commented]():

> Duplicate of https://github.com/code-423n4/2021-12-pooltogether-findings/issues/58

[LSDan (judge) commented]():

> I do not consider this to be a duplicate of 58 because it describes an actual impact that, while extremely unlikely, could result in loss of funds. This also makes it a medium severity issue.

> `2 — Med (M): vulns have a risk of 2 and are considered "Medium" severity when assets are not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

PierrickGT (PoolTogether) confirmed and resolved:

> Fixed by casting to `uint64`. Relevant code: https://github.com/pooltogether/v4-periphery/blob/e0010b689fb170daac77af5f62abba7ca1397524/contracts/TwabRewards.sol#L415-L417 https://github.com/pooltogether/v4-periphery/blob/master/contracts/interfaces/ITwabRewards.sol#L26-L28

[M-05] Missing Check When Transferring Tokens Out For A Given Promotion

Submitted by leastwood

Impact

The `claimRewards` function is called upon by ticket holders who parse a set of `_epochIds` they wish to claim rewards on. An internal call is made to `_calculateRewardAmount` to calculate the correct reward amount owed to the user. Subsequently, the `_updateClaimedEpoch` function will set the epoch bit of the tracked `_claimedEpochs` mapping, ensuring an `epochId` cannot be claimed twice for a given promotion.

However, there may be inaccuracies in the `_calculateRewardAmount` function, which results in more tokens being sent out than allocated by a promotion creator. This severely impacts the ability for users to claim their owed tokens on other promotions.

Proof of Concept

https://github.com/pooltogether/v4-periphery/blob/b520faea26bcf60371012f6cb246aa149abd3c7d/contracts/TwabRewards.sol#L162-L191

```
    function claimRewards(
        address _user,
        uint256 _promotionId,
        uint256[] calldata _epochIds
    ) external override returns (uint256) {
        Promotion memory _promotion = _getPromotion(_promotionId);

        uint256 _rewardsAmount;
        uint256 _userClaimedEpochs = _claimedEpochs[_promotionId][_u

        for (uint256 index = 0; index < _epochIds.length; index++) {
            uint256 _epochId = _epochIds[index];

            require(
                !_isClaimedEpoch(_userClaimedEpochs, _epochId),
                "TwabRewards/rewards-already-claimed"
            );

            _rewardsAmount += _calculateRewardAmount(_user, _promoti
            _userClaimedEpochs = _updateClaimedEpoch(_userClaimedEpo
        }

        _claimedEpochs[_promotionId][_user] = _userClaimedEpochs;

        _promotion.token.safeTransfer(_user, _rewardsAmount);

        emit RewardsClaimed(_promotionId, _epochIds, _user, _rewards

        return _rewardsAmount;
    }
```

Recommended Mitigation Steps

Consider checking that the total rewards claimed for a given promotion is strictly `<=` than the total allotted balance provided by the promotion creator. This should help prevent a single promotion from affecting the rewards claimable from other promotions.

PierrickGT (PoolTogether) acknowledged:

> This check seems redundant, especially now that we have restricted the contract to only support one ticket (in this PR), this will avoid a promotion with a fake ticket to manipulate the amount of rewards claimable and an attacker won't be able to drain funds from a promotion. So for this reason, I've acknowledged the issue but we won't implement this check.

Low Risk Findings (9)

- [L-01] _getPromotion() doesn't revert on invalid _promotionId *Submitted by johnnycash, also found by harleythedog*
- [L-02] cancelPromotion() Does Not Send Promotion Tokens Back to the Creator *Submitted by MetaOxNull*
- [L-03] `getCurrentEpochId()` Malfunction for ended promotions *Submitted by WatchPug, also found by certora*
- [L-04] _requirePromotionActive allows actions before the promotion is active *Submitted by certora*
- [L-05] `getRewardsAmount` might return wrong result *Submitted by certora*
- [L-06] Anyone can claim rewards on behalf of someone *Submitted by cmichel*
- [L-07] Inconsistent definition of when an epoch ends *Submitted by harleythedog*
- [L-08] event PromotionCancelled should also emit the _to address *Submitted by hubble*
- [L-09] No sanity checks for user supplied promotion values *Submitted by kenzo, also found by pauliax and WatchPug*

Non-Critical Findings (2)

- [N-01] extendPromotion function should be access controlled by using onlyPromotionCreator *Submitted by hubble, also found by 0x0x0x and pauliax*
- [N-02] User can fund other user promotion by mistake *Submitted by csanuragjain*

Gas Optimizations (15)

- [G-01] `_nextPromotionId/_latestPromotionId` calculation can be done more efficiently *Submitted by 0x0x0x*
- [G-02] Implement _calculateRewardAmount more efficiently *Submitted by 0x0x0x*
- [G-03] TwarbRewards: don't use the onlyPromotionCreator modifier to save gas *Submitted by GiveMeTestEther, also found by WatchPug*
- [G-04] Check Zero Address Before Function Call Can Save Gas *Submitted by Meta0xNull, also found by Jujic and WatchPug*
- [G-05] Adding unchecked directive can save gas *Submitted by WatchPug, also found by defsec*
- [G-06] `_requireTicket()` Implementation can be simpler and save some gas *Submitted by WatchPug*
- [G-07] simplify require in _requirePromotionActive() *Submitted by gpersoon, also found by 0xabc, WatchPug, cmichel, danb, gzeon, pauliax, pmerkleplant, and sirhashalot*
- [G-08] Inline functions _updateClaimedEpoch and _isClaimedEpoch *Submitted by pauliax*
- [G-09] Struct packing *Submitted by robee, also found by gzeon and leastwood*
- [G-10] Short the following require messages *Submitted by robee, also found by GiveMeTestEther, Jujic, Meta0xNull, WatchPug, defsec, sirhashalot, and yeOlde*
- [G-11] Caching array length can save gas *Submitted by robee, also found by 0x0x0x, defsec, leastwood, and pmerkleplant*
- [G-12] Prefix increments are cheaper than postfix increments *Submitted by robee, also found by Jujic, WatchPug, and defsec*
- [G-13] uint256 types can be uint64 *Submitted by sirhashalot, also found by certora*
- [G-14] Avoid unnecessary dynamic size array _averageTotalSupplies can save gas *Submitted by WatchPug*
- [G-15] Transfer amounts not checked for > 0 *Submitted by yeOlde, also found by WatchPug*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top