PoolTogether TWAB Delegator contest Findings & Analysis Report

2022-04-19

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 audit 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 audit contest outlined in this document, C4 conducted an analysis of the PoolTogether TWAB Delegator smart contract system written in Solidity. The audit contest took place between February 22—February 24 2022.

Wardens

29 Wardens contributed reports to the PoolTogether TWAB Delegator contest:

1. Omik
2. cmichel
3. kirk-baird
4. WatchPug (jtp and ming)
5. CertoraInc (danb, egjlmn1, OriDabush, ItayG, and shakedwinder)
6. hickuphh3

7. Dravee
8. gzeon
9. Rhynorater
10. robee
11. chunter
12. nascent (brock, 0xAndreas, and chris_nascent)
13. jayjonah8
14. lllllll
15. Tomio
16. sorrynotsorry
17. kenta
18. rfa
19. z3s
20. 0x1f8b
21. yeOlde
22. pedroais

This contest was judged by 0xleastwood.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded 1 unique MEDIUM severity vulnerability. Additionally, the analysis included 9 reports detailing issues with a risk rating of LOW severity or non-critical as well as 17 reports recommending gas optimizations. All of the issues presented here are linked back to their original finding.

Notably, 0 vulnerabilities were found during this audit contest that received a risk rating in the category of HIGH severity.

Scope

The code under review can be found within the C4 PoolTogether TWAB Delegator contest repository, and is composed of 4 smart contracts written in the Solidity programming language and includes 420 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/non-critical.

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.

Medium Risk Findings (1)

[M-01] `permitAndMulticall()` May Be Used to Steal Funds Or as a Denial Of Service if `_from` Is Not The Message Sender

Submitted by kirk-baird, also found by cmichel and Omik

Line References

PermitAndMulticall.sol#L46-L64
PermitAndMulticall.sol#L31-L37
TWABDelegator.sol#L438-L445

Vulnerability details

When the `_from` address is not the `msg.sender` `_multiCall()` will be made on behalf of the `msg.sender`. As a result each of the functions called by `multiCall()` will be made on behalf of `msg.sender` and not `_from`.

If functions such as `transfer()` or `unstake()` are called `msg.sender` will be the original caller which would transfer the attacker the funds if the `to` field is set to an attackers address.

Furthermore, if an attacker we to call `permitAndMulticall()` before the `_from` user they may use their signature and nonce combination. As a nonce is only allowe to be used once the siganture will no longer be valid and `_permitToken.permit()` will fail on the second call.

An attacker may use this as a Denial of Service (DoS) attack by continually front-running `permitAndCall()` using other users signatures.

Proof of Concept

```
function _multicall(bytes[] calldata _data) internal virtual ret
  results = new bytes[](_data.length);
  for (uint256 i = 0; i < _data.length; i++) {
    results[i] = Address.functionDelegateCall(address(this), _da
  }
  return results;
}
```

```
function _permitAndMulticall(
  IERC20Permit _permitToken,
  address _from,
  uint256 _amount,
  Signature calldata _permitSignature,
  bytes[] calldata _data
) internal {
  _permitToken.permit(
    _from,
    address(this),
    _amount,
```

```
        _permitSignature.deadline,
        _permitSignature.v,
        _permitSignature.r,
        _permitSignature.s
      );

      _multicall(_data);
    }
```

Recommended Mitigation Steps

Consider updating the `_from` field to be the `msg.sender` in `permitAndMulticall()` (or alternatively do this in `_permitAndMulticall()` to save some gas).

```
    function permitAndMulticall(
      uint256 _amount,
      Signature calldata _permitSignature,
      bytes[] calldata _data
    ) external {
      _permitAndMulticall(IERC20Permit(address(ticket)), msg.sender,
    }
```

PierrickGT (PoolTogether) confirmed and resolved:

> PR: https://github.com/pooltogether/v4-twab-delegator/pull/29

PierrickGT (PoolTogether) disagreed with Medium severity and commented:

> Should be labelled as a 3 (High Risk) issue because an attacker could steal the funds.

Oxleastwood (judge) commented:

> I'm not exactly sure how this might be abused to steal funds. By front-running a call `permitAndMulticall()` with the same `_from` account, an attacker is able to use up the user's nonce and DoS their transactions. However, an attacker CAN control the `_data` parsed to the `_multicall()` function and delegate call to the

> `TWABDelegator.sol` contract. Although, in this case `msg.sender` will be the attacker and not the delegatee.

> As such, any call to transfer a delegation to another account will fail as the delegation is computed based off `msg.sender` and `_slot`.

> Could you confirm if there is a viable attack vector that would result in lost funds? @PierrickGT

PierrickGT (PoolTogether) commented:

> You are right, the only attack vector possible would be with the `updateDelegatee` function since an attacker could pass a `_delegatee` address and we compute the delegation with the passed `_delegator` param. https://github.com/pooltogether/v4-twab-delegator/blob/60ae14e11947f8c896c1fef8f4d19ee714719383/contracts/TWABDelegator.sol#L265

> Funds wouldn't be at risk but delegated to the attacker address. So I think the `2 (Med Risk)` label makes sense in this case since funds are not directly at risk but the attacker would enjoy better odds of winning. I've removed the `disagree with severity` label.

0xleastwood (judge) commented:

> As per the above comment, I will leave this as `2 (Med Risk)`. The exploit does not lead to a loss of funds but can be abused to DoS this functionality and enjoy better odds of winning.

Low Risk and Non-Critical Issues

For this contest, 9 reports were submitted by wardens detailing low risk and non-critical issues. The reports highlighted below received the top 3 scores from the judge and are by the following wardens/teams: 1) WatchPug; 2) CertoraInc; and 3) hickuphh3.

The following wardens also submitted reports: gzeon, Rhynorater, Dravee, chunter, jayjonah8, *and* robee.

[L-01] delegator and/or representative should be allowed for arbitrary code execution besides restricted operations during unlocked period

Submitted by WatchPug

Line References

[Delegation.sol#L39-L46]

Vulnerability details

`Delegation` is a contract deployed dedicated to holding the `ticket` tokens for the `delegator` and they can then be `delegate` to a `delegatee`.

On the `Delegation` contract, there is a method named `executeCalls()` designed for "Executes calls on behalf of this contract" which allows arbitrary code execution for the owner.

However, we found that the owner of `Delegation` will always be `TWABDelegator`, and the `TWABDelegator` will only use `Delegation.sol#executeCalls()` to call one particular address: the `ticket` address, and for only two methods: `transfer()` and `delegate()`.

Furthermore, even though in `Delegation.sol#executeCalls()`, `calls[i].value` is used, the function is not being marked as `payable`, that makes it hard for calls that requires eth payments.

[Delegation.sol#L39-L46]

```
    function executeCalls(Call[] calldata calls) external onlyOwner
      bytes[] memory response = new bytes[](calls.length);
      for (uint256 i = 0; i < calls.length; i++) {
        response[i] = _executeCall(calls[i].to, calls[i].value, call
      }
      return response;
    }
```

While the `ticket` is being delegated through `TWABDelegator`, they won't be able to retrieve the tickets back until the `lockUntil`, without the ability to make arbitrary code execution, the `delegator` may miss some of the potential benefits as a holder of the `ticket` tokens, for example, an airdrop to all holders of the `ticket` tokens, or an NFT made mintable only for certain ticket holders.

Recommended Mitigation Steps

Consider adding a new method on `TWABDelegator`:

```
    function executeCalls(
      address _delegator,
      uint256 _slot,
      Delegation.Call[] memory calls
    ) external payable returns (bytes[] memory) {
      _requireDelegatorOrRepresentative(_delegator);
      Delegation _delegation = Delegation(_computeAddress(_delegator

      if (block.timestamp < _delegation.lockUntil()) {
        for (uint256 i = 0; i < calls.length; i++) {
          if (calls[i].to == address(ticket)) {
            revert("TWABDelegator/delegation-locked");
          }
        }
      }

      return _delegation.executeCalls{value: msg.value}(_calls);
    }
```

And also, consider making `Delegation.sol#executeCalls()` a `payable` method.

PierrickGT (PoolTogether) acknowledged and commented:

> The issue outlined by the warden is relevant but users won't need to execute arbitrary calls cause potential rewards given out to ticket holders will be handled by our TWABRewards contract. This contract retrieves users TWAB (Time-Weighted Average Balance) for a given period of time and calculate the amount of rewards they are eligible to. Users can then `claimRewards` on behalf of others. So delegatees will be able to claim their rewards and delegators could claim on their behalf.

> TwabRewards.sol#L410
> ITwabRewards.sol#L94

> For more informations about how the TWAB works, here is some documentation:

- Time-Weighted Average Balance
- Better Reward Distribution

> Also, by restricting calls to the `transfer` and delegatè methods on the ticket, we limit the attack surface and any attack vector we may not have thought about.

> For the reasons above, I've acknowledged the issue but we won't implement the proposed solution

0xleastwood (judge) decreased severity to Low and commented:

> I don't really see a case where `_delegateCall()` or `_transferCall()` will need to have some ETH attached with it. They are solely dealing with the Ticket ERC20 token and updating delegation data. Considering the fact that rewards are handled by a separate contract, I think its fair to downgrade this to `1 (Low Risk)`.

[N-01] Tickets can get locked

Submitted by CertoraInc

if a user calls `transferDelegationTo` with the address of the `TWABDelegator` contract as the `to` parameter, the tokens will be transferred to the address without minting the user the stake token, so maybe you can think of adding this functionality. I know that there is the `withdrawDelegationToStake` function for that, but that can be nice to enable it that way too.

PierrickGT (PoolTogether) confirmed, but disagreed with severity and commented:

> The tickets would actually get stuck in the contract. I've added a require to avoid transferring directly to the contract.

> Based on the severity criteria, I think this one should be labelled as a 2 (Med Risk) issue. Would be an error from the user interacting with the functions but funds would indeed be at risk, not direct but at risk.

> PR: pooltogether/v4-twab-delegator#27

Oxleastwood (judge) commented:

> I think the issue highlighted is more in-line with incorrect state handling, so I'll leave the severity as is.

[L-02] Incorrect comment on `transferDelegationTo()`

Submitted by hickuphh3

Line References

TWABDelegator.sol#L370-L371

Description

The comments say that the withdrawn tickets are transferred to the caller / delegator wallet, but are actually transferred to the `_to` address.

Recommended Mitigation Steps

```
    * @notice Withdraw an `_amount` of tickets from a delegation. Th
    * @dev Tickets are sent directly to the passed `_to` address
```

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator#21

[L-03] Incorrect comment on `TransferredDelegation` event

Submitted by hickuphh3

Line References

[TWABDelegator.sol#L125-L136]

Description

In relation to L03, the `TransferredDelegation` event

```
- is incorrectly commented that the withdrawn tickets are transf
- lacks a description about the `to` indexed parameter.
```

Recommended Mitigation Steps

```
/**
 * @notice Emitted when a delegator withdraws an amount of ticke
 * @param delegator Address of the delegator
 * @param slot  Slot of the delegation
 * @param amount Amount of tickets withdrawn
 * @param to Recipient address of withdrawn tickets
 */
  event TransferredDelegation(
    address indexed delegator,
    uint256 indexed slot,
    uint256 amount,
    address indexed to
  );
```

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator#21

[L-04] Incorrect comment on `DelegationFundedFromStake` event

Submitted by hickuphh3

Line References

TWABDelegator.sol#L102

Description

The `DelegationFundedFromStake()` allows a representative or delegator himself to fund a delegation contract using the delegator's stake. The `user` in the `DelegationFundedFromStake` event refers to `msg.sender`. Since the funds are coming solely from the delegator, its description isn't entirely correct.

Recommended Mitigation Steps

```
@param user Address of the user who pulled funds from the delegator to the delegation
```

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator#21

[N-02] Extra whitespace in slot description of `WithdrewDelegationToStake()` event

Submitted by hickuphh3

Line References

TWABDelegator.sol#L114

Description

There is an additional spacing between `slot` and `Slot`.

Recommended Mitigation Steps

Remove the spacing to become: `* @param slot Slot of the delegation`

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator#21

[N-03] TWABDelegator: Consider renaming `_delegateCall()` to `_setDelegateeCall()`

Submitted by hickuphh3

Line References

[TWABDelegator.sol#L519](#)

Description

`_delegateCall()` could easily be confused for the inbuilt `delegatecall()` method. I recommend renaming it to something more distinguishable like `_setDelegateeCall()`.

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator#21

Gas Optimizations

For this contest, 17 reports were submitted by wardens detailing gas optimizations. The report highlighted below by warden team **CertoraInc** received the top score from the judge.

The following wardens also submitted reports: Dravee, nascent, IIIIIII, WatchPug, robee, Tomio, sorrynotsorry, kenta, gzeon, rfa, z3s, Omik, 0x1f8b, ye0lde, pedroais, *and* hickuphh3.

[G-01] Loop in `Delegation` and `PermitAndMulticall` contracts

Loops can be optimized in several ways. Let's take for example the loop in the `executeCalls` function of the `Delegation` contract.

```
function executeCalls(Call[] calldata calls) external onlyOwner
```

```
    bytes[] memory response = new bytes[](calls.length);
    for (uint256 i = 0; i < calls.length; i++) {
      response[i] = _executeCall(calls[i].to, calls[i].value, call
    }
    return response;
  }
```

To optimize this loop and make it consume less gas, we can do the foloowing things:

1. Use ++i instead of i++, which is a cheaper operation (in this case there is no difference between i++ and ++i because we dont use the return value of this expression, which is the only difference between these two expression).
2. Save the `calls` array length in a local variable instead of accessing it in every iteration.
3. Save `calls[i]` in a local variable instead of accessing it 3 times in every iteration. This will save accssing the array's ith element 3 times in every iteration ,which requires an address calculation.
4. There's no need to initialize i to its default value, it will be done automatically and it will consume more gas if it will be done (I know, sounds stupid, but trust me - it works).

So after applying all these changes, the loop will look something like this:

```
function executeCalls(Call[] calldata calls) external onlyOwner
  bytes[] memory response = new bytes[](calls.length);
  uint256 length = calls.length;
  Call memory call;
  for (uint256 i; i < length; ++i) {
    call = calls[i];
    response[i] = _executeCall(call.to, call.value, call.data);
  }
  return response;
}
```

[G-02] Inline all these little functions

Defining all these little functions cause 2 things:

1. contract's code size gets bigger
2. the function calls consumes more gas than exectuing it as an inlined function (part of the code, without the function call)

So in order to save gas, I would recommend to inline these functions.

```
function _computeAddress(address _delegator, uint256 _slot) inte
  return _computeAddress(_computeSalt(_delegator, bytes32(_slot)
}

function _computeLockUntil(uint96 _lockDuration) internal view r
  return uint96(block.timestamp) + _lockDuration;
}

function _requireDelegatorOrRepresentative(address _delegator) i
  require(
    _delegator == msg.sender || representatives[_delegator][msg.
    "TWABDelegator/not-delegator-or-rep"
  );
}

function _requireDelegateeNotZeroAddress(address _delegatee) int
  require(_delegatee != address(0), "TWABDelegator/dlgt-not-zerc
}

function _requireAmountGtZero(uint256 _amount) internal pure {
  require(_amount > 0, "TWABDelegator/amount-gt-zero");
}

function _requireDelegatorNotZeroAddress(address _delegator) int
  require(_delegator != address(0), "TWABDelegator/dlgtr-not-zer
}

function _requireRecipientNotZeroAddress(address _to) internal p
  require(_to != address(0), "TWABDelegator/to-not-zero-addr");
}

function _requireDelegationUnlocked(Delegation _delegation) inte
  require(block.timestamp >= _delegation.lockUntil(), "TWABDeleg
}

function _requireContract(address _address) internal view {
  require(_address.isContract(), "TWABDelegator/not-a-contract")
}
```

```
function _requireLockDuration(uint256 _lockDuration) internal pu
  require(_lockDuration <= MAX_LOCK, "TWABDelegator/lock-too-lor
}
```

PierrickGT (PoolTogether) confirmed and commented:

> PR: pooltogether/v4-twab-delegator/pull#18

> We've implemented the different fixes regarding the for loops, except for the `++i` recommendation, we kept `i++` for better code clarity. About the inline suggestion, we prefer to keep the code in reusable functions to keep a more readable and easier to update codebase than if we had to repeat our code through inlining.

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