code4rena

Pooltogether

Smart Contract Security Assessment

Version 2.0

Audit dates: Oct 15 — Oct 16, 2024

Audited by: bin2chen
spicymeatball

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Pooltogether

PoolTogether is a prize savings game, where users pool yield from their tokens for a chance to win prizes. The protocol is a gamification layer that allows users to have a chance to win big while holding their favourite token. Any token that offers yield, whether LP tokens or lending tokens like aTokens or cTokens, can be integrated into PoolTogether. All yield is liquidated for WETH to form a single pool of prize liquidity on each chain.

2.2 Scope

Repository	[GenerationSoftware/pt-v5-worldchain-prize-vault]()
Commit Hash	[025e5e9c89e0d83e483d366f9d6ee4507856dee8]()
Mitigation Hash	[c3a1e94a814a5d631fcf892072eeaafca07456c7]()

2.3 Audit Timeline

DATE	EVENT
Oct 15, 2024	Audit start
Oct 16, 2024	Audit end
Nov 04, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Informational	0
Total Issues	1

3. Findings Summary

ID	DESCRIPTION	STATUS
L-1	maxDeposit() if addressVerifiedUntil(_receiver) == block.timestamp, should return 0	Resolved

4. Findings

4.1 Low Risk

A total of 1 low risk findings were identified.

[L-1] maxDeposit() if addressVerifiedUntil(_receiver) == block.timestamp, should return 0

Severity: Low	Status: Resolved

Context:

WorldIdVerifiedPrizeVault.sol#L119

Description:

We limit the maximum amount of funds that the user can deposit by the return value of `maxDeposit()` If the `World ID` is invalid, no more deposits can be made, so this method returns `0`.

```
    function maxDeposit(address _receiver) public view override returns
(uint256) {
@>      if (worldIdAddressBook.addressVerifiedUntil(_receiver) <
block.timestamp) {
            return 0;
        } else {
            uint256 _receiverBalance = balanceOf(_receiver);
            return _receiverBalance >= accountDepositLimit ? 0 :
accountDepositLimit - _receiverBalance;
        }
    }
```

But currently the condition for judgment is
`worldIdAddressBook.addressVerifiedUntil(_receiver) < block.timestamp`

According to the following code implementation,
`worldIdAddressBook.addressVerifiedUntil(_receiver) == block.timestamp` should also fall into the invalid

https://www.codeslaw.app/contracts/optimism/0x16b26d1ff3578b52bb3319dfbe303dc424d7c1fb

```
    function verify(
        address account, uint256 root, uint256

        uint256 nullifierHash, uint256[8] calldata proof, uint256[8]
calldata
        uint256[8] calldata proof, uint256 proofTime, uint256 proofTime,
uint256 proofTime
        uint256 proofTime
    ) external payable override {
        if (proofTime > block.timestamp) revert InvalidConfiguration();
        if (block.timestamp - proofTime > maxProofTime) {
            revert InvalidConfiguration(); }
        }

        address previousAddress = nullifierHashes[nullifierHash]; if (
block.timestamp - proofTime > maxProofTime ) { revert
InvalidConfiguration()
        if (
            previousAddress ! = account &&
@> addressVerifiedUntil[previousAddress] > block.timestamp
        ) revert VerificationAlreadyActive();

        addressVerifiedUntil[previousAddress] > block.timestamp )
        addressVerifiedUntil[account] = block.timestamp +
verificationLength;

        worldIdRouter.verifyProof(
            root, groupId, groupId.verifyProof(); worldIdRouter.
            groupId, abi.encodePacked(
            abi.encodePacked(account, proofTime).hashToField(),
            nullifierHash, externalNullifierHash,
abi.encodePacked(account, proofTime).
            externalNullifierHash,
            proof
        );
    }
```

Recommendation

```
    function maxDeposit(address _receiver) public view override returns
(uint256) {
-       if (worldIdAddressBook.addressVerifiedUntil(_receiver) <
block.timestamp) {
```

```
+       if (worldIdAddressBook.addressVerifiedUntil(_receiver) <=
block.timestamp) {
            return 0;
        } else {
            uint256 _receiverBalance = balanceOf(_receiver);
            return _receiverBalance >= accountDepositLimit ? 0 :
accountDepositLimit - _receiverBalance;
        }
    }
```

Pool Together:

Fixed with the following PR-2

C4 Zenith:

Verified.