SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	PoolTogether
Prepared by:	Sherlock
Lead Security Expert:	hash
Dates Audited:	May 16 - June 6, 2024
Prepared on:	July 15, 2024

Introduction

PoolTogether is a prize savings protocol. It makes holding your favourite token exciting by gamifying yield as prizes!

Scope

Repository: GenerationSoftware/pt-v5-twab-controller

Branch: main

Commit: 827255118b0de751bc797de6bf6ed042496aea4d

Repository: GenerationSoftware/pt-v5-vault

Branch: main

Commit: 436b06fbe33d7c4616dea4dbdb262237c1436cb6

Repository: GenerationSoftware/pt-v5-prize-pool

Branch: main

Commit: 768fa642eb31cfff0fe929da0929a9bb4dea0b2d

Repository: GenerationSoftware/pt-v5-draw-manager

Branch: main

Commit: f04edd938f0ce3d6bbaf5db2748319d6ebf6b078

Repository: GenerationSoftware/pt-v5-rng-witnet

Branch: main

Commit: ac310b9deb1e53a547e53d69861495888d322ac3

Repository: GenerationSoftware/pt-v5-claimer

Branch: main

Commit: a3619aa13c19beb25210ddb6474cd51aac794706

Repository: GenerationSoftware/pt-v5-tpda-liquidator

Branch: main

Commit: 2f7aeb0ebc88a650791e7e56dee33e9981f3ed14

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
19	3

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x73696d616f
hash
MiloTruck
elhaj
0xSpearmint1
jo13
berndartmueller
trachev
infect3d
zraxx
Rhaydden
jovi
volodya
ydlee
Tri-pathi
cu5t0mPe0
aman
0xAadi
AuditorPraise
dany.armstrong90

Issue H-1: Vault portion calculation in `PrizePool::getVaultPortion()` is incorrect as `_startDrawIdInclusive` has been erased

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/96

Found by

0x73696d616f

Summary

The vault portion calculation in `PrizePool::getVaultPortion()` is incorrect because it fetches a `_startDrawIdInclusive` that has been overwritten in the total accumulator but not in the vaults accumulator or the donation accumulator.

Vulnerability Detail

To illustrate the issue, consider the grand prize, where odds are `1 / g = 0.00273972602e18` with `g == 365 days`. The estimated prize frequency is `1e18 / 0.00273972602e18 == 365.000000985` rounded up to the nearest integer, `366`. If `lastAwardedDrawId_` is `366`, then `startDrawIdInclusive` is `366 - 366 + 1 == 1`. This means that it will fetch the total, vault and donation accumulators since the opening of `drawId == 1`.

However, when `lastAwardedDrawId_ == 366`, the current open `drawId` is `367`, which is written to index 0 in the circular buffer, overwriting `drawId == 1`. But, in the donation and vault cases, the accumulator still has `drawId == 1`, as the buffer will likely not have been overwritten (draw periods take 1 day, it is highly likely the individual accumulators are not written to every day, but the total accumulator is, or users may exploit this on purpose). As the buffer is circular, this will keep happening every time after the buffer has been filled and indexes start being overwritten.

Thus, due to this, the vault portion will be bigger than it should, as the total accumulator is calculated between 365 days (from 2 to 366, as 1 has been erased), but the vault and donation accumulators for 366 days (between 1 and 366). Users will have bigger than expected chances of winning prizes and in the worst case, they may even not be able to claim prizes at all, in case the erased `drawId` had a significant contribution to the prizes through a donation, and then it underflows when doing `totalSupply = totalContributed - totalDonated;` in PrizePool::getVaultShares().

Add the following test to `PrizePool.t.sol` and log the vault and total portions in

`PrizePool::getVaultShares()`. The contribution of the vault will be bigger than the total.

```
function testAccountedBalance_POC() public {
  for (uint i = 0; i < 366; i++) {
    contribute(100e18);
    awardDraw(1);
    mockTwab(address(this), msg.sender, 0);
  }
  address otherVault = makeAddr("otherVault");
  vm.prank(otherVault);
  prizePool.contributePrizeTokens(otherVault, 0);
  uint256 prize = claimPrize(msg.sender, 0, 0);
}
```

Impact

Users get better prize chances than supposed and/or it can lead to permanent reverts when trying to claim prizes.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/main/pt-v5-prize-pool/src/PrizePool.sol#L1014

Tool used

Manual Review

Vscode

Recommendation

When calculating `PrizePool::computeRangeStartDrawIdInclusive()`, the `rangeSize` should be capped to `365` to ensure the right `startDrawIdInclusive` is fetched, not one that has already been erased. In the scenario above, if the range is capped at 365, when `lastAwardedDrawId_ == 366`, `startDrawIdInclusive` would be `366 - 365 + 1 == 2`, which has not been erased as the current `drawId` is `367`, having only overwritten `drawId == 1`, but 2 still has the information.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-prize-pool/pull/115

10xhash

Fixed. Now the max range of draws to look for is limited to `grandPrizePeriodDraws`

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-2: Unfair Manipulation of Winning Chances Due to Stolen Yield on `Blast`

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/114

Found by

0xSpearmint1, elhaj, jo13

Summary

- The `PoolTogether` protocol on Blast will use `WETH` as it's prize token as mentioned by sponsor, which generates yield over time. This yield can be stolen and used to manipulate the vaults' winning chances unfairly. By contributing the stolen yield to their own vault, users can inflate their vault's portion, increasing their chances of winning prizes while decreasing the chances for other legitimate users. This undermines the fairness of the prize distribution.

Vulnerability Detail

- The `PoolTogether` protocol will be deployed on Blast, and as the sponsor mentioned, `WETH` will be the prize token.
- `WETH` on Blast is a rebasing token that automatically generates yield over time. `WETH` holders have three Yield Modes for rebasing: `Void`, `Automatic`, and `Claimable`. The **Default** mode is `Automatic`, which means the balance will increase over time by the yield generated.
- The prizePool will hold `WETH` as the prize token and will generate yield over time. However, this yield can be stolen by anyone, as they can contribute it to their own vault by calling the `contributePrizeTokens()` function, passing their vault and the amount of yield generated.

```
function contributePrizeTokens(address _prizeVault, uint256 _amount) public
↪  returns (uint256) {
  >>  uint256 _deltaBalance = prizeToken.balanceOf(address(this)) -
↪  accountedBalance();
    if (_deltaBalance < _amount) {
      revert ContributionGTDeltaBalance(_amount, _deltaBalance);
    }
    uint24 openDrawId_ = getOpenDrawId();
   >> _vaultAccumulator[_prizeVault].add(_amount, openDrawId_);
   >> _totalAccumulator.add(_amount, openDrawId_);
```

```
    emit ContributePrizeTokens(_prizeVault, openDrawId_, _amount);
    return _deltaBalance;
  }
```

- The idea is that this yield is generated from all vault contributions. However, a malicious user can exploit this feature to their advantage. While this might be considered a missing feature (not benefiting from yield) in different protocol contexts, within the `PoolTogether` on Blast scenario, it has a detrimental effect. The malicious user's ability to exploit the yield generation artificially inflates their vault's portion compared to other vaults. **This act will decrease other vaults portions consequently increases the malicious user vault portion, thus increase his chances of winning prizes and decrease others chances**.
- For the malicious user's vault, the increase in their contribution portion will lead to a higher `winning zone` thus more chances to win. Conversely, for other vaults, the decrease in their contribution portions will lead to a smaller `winning zone` reducing their chances of winning.
- This manipulation undermines the fairness of the prize distribution. Not only will the yield be stolen from all vaults contributions and not be spread to them proportionally, but it will also affect their winning chances by decreasing them. This gives the malicious user an unfair advantage by increasing their likelihood of winning at the expense of others.

Scenario

Normal Case

- Let's say from draw 4 to draw 40, we have:
  - `vault1.contributionBetween = 100`
  - `vault2.contributionBetween = 100`
  - `totalContributionBetween = 200`
- To calculate their portions:
  - `v1.portion = 100 / 200 = 0.5`
  - `v2.portion = 100 / 200 = 0.5`

  Assuming the TWAB balance is 500 for both vaults and they both have 1 user for easy calculation(so users twab also 500) , and ignoring the odds scaling:
- `winningZone(vault1[user]) = 500 * 0.5 = 250`
- `winningZone(vault2[user]) = 500 * 0.5 = 250`

Yield Stolen Case

- Now let's assume that a malicious user continuously steals yield during this period (from draw 4 to draw 40)

  `Notice` that the yield will be generated for all the PrizePool balance (probably more than 200 which totalContribution in this period):

- When we get the contribution of `vault3` (malicious vault) for this period (from draw 4 to 40), we get:
  - `vault3.contribution = 50`
- Now let's recalculate the vaults' portions again:
  - `vault1.contributionBetween = 100`
  - `vault2.contributionBetween = 100`
  - `vault3.contributionBetween = 50`
  - `totalContributionBetween = 250`
- To calculate their portions:
  - `v1.portion = 100 / 250 = 0.4`
  - `v2.portion = 100 / 250 = 0.4`
  - `v3.portion = 50 / 250 = 0.2`

  Assuming the TWAB balance is 500 for all vaults:

- `winningZone(vault1[user]) = 500 * 0.4 = 200`
- `winningZone(vault2[user]) = 500 * 0.4 = 200`
- `winningZone(vault3[maliciousUser) = 500 * 0.2 = 100`
- Notice how the malicious user now has a chance to win without any contribution from it's vault (only stolen yield), and how legitimate users vaults' chances have decreased.

Impact

- Malicious actors can steal yield generated by the prize pool, inflating their vault's portion and unfairly increasing their win chances. This decreases legitimate players' portion and winning probabilities.

Code Snippet

- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L1037

-L1043

- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L412-L422

Tool used

manual review

Recommendation

- For `Blast`, configure the yield mode to `Claimable` in the constructor so it doesn't affect the `balanceOf()`. Expose a public function that claims this yield and contributes it to the `DONATOR` address. This way, the yield is not wasted, and all contributors benefit from it. Contributors will be incentivized to call this function since it increases the prize size.

```
 // import those :
 enum YieldMode {
  AUTOMATIC,
  VOID,
  CLAIMABLE
}
interface IERC20Rebasing {
  function configure(YieldMode) external returns (uint256);
  function claim(address recipient, uint256 amount) external returns (uint256);
  function getClaimableAmount(address account) external view returns (uint256);
}
```

```
contract prizePool {

 constructor() {

   // you can check by address of by chainid if you are in blast :
+  if (address(prizeToken) == 0x4300000000000000000000000000000000000004){
+        IERC20Rebasing(address(prizeToken)).configure(YieldMode.CLAIMABLE);
 ↪  //configure claimable yield for WETH
+    }
+  }

+ function claimYield() external returns (uint256){
+    IERC20Rebasing weth = IERC20Rebasing(address(prizeToken));
+    uint amount = weth.getClaimableAmount(address(this));
+     weth.claim(address(this),amount);
```

```
+      contributePrizeTokens(DONATOR, amount);
+  }
}
```

Discussion

trmid

The blast docs here state that:

> Smart contract accounts have three Yield Modes for their rebasing mode:
>
> - Void (DEFAULT): ETH balance never changes; no yield is earned
> - Automatic: native ETH balance rebases (increasing only)
> - Claimable: ETH balance never changes; yield accumulates separately

The docs say the default for EOAs is rebasing, but the default for the contracts like the prize pool will be void.

elhajin

@trmid that's not true that's the case for native ETH not WETH here a quote from docs in case of WETH:

> Similar to ETH, WETH and USDB on Blast is also rebasing and follows the same yield mode configurations. **However, unlike ETH where contracts have Disabled yield by default, WETH and USDB accounts have Automatic yield by default for both EOAs and smart contracts**

nevillehuang

@elhajin You are right based on this link here. Seems like this is a valid issue.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-prize-pool/pull/114

10xhash

Fixed Now WETH is configured claimable and is donated

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-3: Draw auction rewards likely exceed the available rewards, resulting in overpaying rewards or running into an `InsufficientReserve` error

Source:
https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/125

Found by

0x73696d616f, MiloTruck, berndartmueller, hash, infect3d, trachev, zraxx

Summary

The sum of the calculated `startDraw` and `finishDraw` reward fractions likely exceeds `1.0`, which can lead to overpaying rewards or an `InsufficientReserve` error.

Vulnerability Detail

The `finishDraw` function calculates the rewards for starting and finishing a draw and distributes them to the respective participants. The total rewards (`availableRewards`) are based on the available prize pool reserve (upper capped by maxRewards). Those rewards are then split into the start draw rewards and the finish draw rewards. Internally, the portion each participant receives is denominated in fractions ([0.00, 1.00]). No more than `availableRewards` (i.e., the upper cap or the total available prize pool reserve)can be distributed.

However, it is very likely that the sum of all fractions exceeds `1.0`, which would result in utilizing more rewards than `availableRewards`. Consequently, if the reserve has been larger than the `maxRewards` upper cap, more rewards would be paid out than `availableRewards`. Or, if the reserve has been smaller than `maxRewards`, which means the entire reserve is supposed to get used for rewards, using more than `1.0 * availableRewards` would result in an InsufficientReserve error.

Basically, in the former case, rewards are overpaid, and in the latter case, the draw can not be awarded due to the `InsufficientReserve` error.

The following test case demonstrates this issue by having two `startDraw` calls, both of which are supposed to receive `~0.1e18` rewards, which would already exceed the `0.1e18` available reserve. Please note that due to mocking the calls to the prize pool, the actual `InsufficientReserve` error is not triggered. However, by inspecting the emitted events, it can be seen that the rewards are overpaid.

```
function test_finishDraw_multipleStarts_with_exceeding_reserve() public {
    vm.warp(1 days + auctionDuration);
```

```
    mockReserve(.1e18, 0);

    // 1st start draw
    mockRng(99, 0x1234);
    uint firstStartReward = 99999999999999999; // ~0.1e18
    vm.expectEmit(true, true, true, true);
    emit DrawStarted(address(this), alice, 1, auctionDuration, firstStartReward,
↪   99, 1);
    drawManager.startDraw(alice, 99);

    // RNG fails
    mockRngFailure(99, true);

    // 2nd start draw
    vm.warp(1 days + auctionDuration + auctionDuration);
    vm.roll(block.number + 1);
    mockRng(100, 0x1234);
    uint secondStartReward = 99999999999999999; // ~0.1e18 - Would already
↪   exceed the available reserve of 0.1e18
    vm.expectEmit(true, true, true, true);
    emit DrawStarted(address(this), bob, 1, auctionDuration, secondStartReward,
↪   100, 2);
    drawManager.startDraw(bob, 100);

    mockFinishDraw(0x1234);
    vm.warp(1 days + auctionDuration*3);
    uint finishReward = 99999999999999999; // ~0.1e18
    mockAllocateRewardFromReserve(alice, firstStartReward);
    mockAllocateRewardFromReserve(bob, secondStartReward);
    mockAllocateRewardFromReserve(bob, finishReward);
    mockReserveContribution(.1e18);
    assertEq(drawManager.finishDrawReward(), finishReward, "finish draw reward
↪   matches");
    drawManager.finishDraw(bob);
}
```

Please copy and paste the above test case into the `pt-v5-draw-manager/test/DrawManager.t.sol` file and run it with `forge test -vv --match-test "test_finishDraw_multipleStarts_with_exceeding_reserve"`.

Impact

Either rewards for `startDraw` and `finishDraw` are overpaid, or the draw can not be awarded.

Code Snippet

DrawManager.finishDraw#L314-L352

```
314: function finishDraw(address _rewardRecipient) external returns (uint24) {
...    // [...]
333:   uint256 availableRewards = _computeAvailableRewards();
334:   (uint256[] memory startDrawRewards, UD2x18[] memory startDrawFractions) =
↪  _computeStartDrawRewards(
335:     prizePool.drawClosesAt(startDrawAuction.drawId),
336:     availableRewards
337:   );
338:   (uint256 _finishDrawReward, UD2x18 finishFraction) =
↪  _computeFinishDrawReward(
339:     startDrawAuction.closedAt,
340:     block.timestamp,
341:     availableRewards
342:   );
343:   uint256 randomNumber = rng.randomNumber(startDrawAuction.rngRequestId);
344:   uint24 drawId = prizePool.awardDraw(randomNumber);
345:
346:   lastStartDrawFraction = startDrawFractions[startDrawFractions.length - 1];
347:   lastFinishDrawFraction = finishFraction;
348:
349:   for (uint256 i = 0; i < _startDrawAuctions.length; i++) {
350:     _reward(_startDrawAuctions[i].recipient, startDrawRewards[i]);
351:   }
352:   _reward(_rewardRecipient, _finishDrawReward);
```

Tool used

Manual Review

Recommendation

Consider ensuring that the sum of all fractions does not exceed `1.0` to prevent overpaying rewards or running into an `InsufficientReserve` error.

Discussion

trmid

The suggested mitigation is a good addition to the draw auction logic, however the difference in behaviour between the fix and the current behaviour is minimal:

Current Behaviour:

When there is not enough reserve to pay for the `finishDraw`, the draw will not be awarded and the prize liquidity will gracefully roll over to the next draw (the prize pool is well-equipped to handle skipped draws). The only loss here is on the agent that started the draw, who will not receive the rewards for doing so.

Behaviour After Fix

When there is not enough reserve to pay for the `finishDraw`, the auction price will remain at the value of whatever is left until the end of the auction incase a benevolent actor decides to push it through at a loss. If no one pushes it through, the draw will be skipped the same as before and the agent that started the draw will still receive no rewards.

The only difference is the amount of time that a benevolent actor has to push the draw through at a loss. No core protocol functionality is broken since the prize pool is only expected to award prizes if there are enough incentives to do so, and it's built to handle skipped draws without loss.

nevillehuang

@trmid I see users that was supposed to be awarded in the original draw as losing their funds as well (since they lose their chance to win), not just the agent that started the draw.

Was this behavior documented as expected behavior? If not I would lean towards agreeing that high severity is appropriate.

trmid

A severity of `high` seems inappropriate for this issue.

The prize pool uses part of the yield generation (the reserve) to incentivize the RNG auctions. If the generated value is insufficient for a draw to be awarded, then the prize pool keeps all the prize liquidity for the draw and includes it in the next draw. The issue mentioned here is not something that can be forced or exploited, rather it is describing a better way to handle the auction pricing when there is less yield than required to incentivize the RNG. The only difference between the current behaviour and the suggested mitigation behaviour is that the auction will be available for a longer period of time when it does not have proper liquidity for a profitable payout.

This issue is assuming that there are not enough incentives for the RNG auctions to provide profitable payouts, therefore `medium` seems more appropriate:

> Causes a loss of funds but requires certain external conditions or specific states, or a loss is highly constrained. The losses must exceed small, finite amount of funds, and any amount relevant based on the precision or significance of the loss.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/GenerationSoftware/pt-v5-draw-manager/pull/18

10xhash

Fixed with https://github.com/GenerationSoftware/pt-v5-draw-manager/pull/17 and https://github.com/GenerationSoftware/pt-v5-draw-manager/pull/18 Now the max reward is limited to remaining fraction after the other rewards are considered

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-1: Draws can be retried even if a random number is available or the current draw has finished

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/27

Found by

MiloTruck

Summary

`RngWitnet.isRequestFailed()` does not accurately represent whether a random number is available, making it possible for `startDraw()` to be called in conditions that causes a loss of funds for draw bots.

Vulnerability Detail

`RngWitnet.isRequestFailed()` determines if a request has failed by checking the response status of a specific block number is equal to `WitnetV2.ResponseStatus.Error`:

RngWitnet.sol#L115-L118

```
function isRequestFailed(uint32 _requestId) onlyValidRequest(_requestId) public
↪  view returns (bool) {
    (uint256 witnetQueryId,,) =
↪  witnetRandomness.getRandomizeData(requests[_requestId]);
    return witnetRandomness.witnet().getQueryResponseStatus(witnetQueryId) ==
↪  WitnetV2.ResponseStatus.Error;
}
```

Note that `requests[_requestId]`, stores the block number at which the request at `_requestId` was made.

However, `isRequestFailed()` does not accurately reflect if a random number is available. In Witnet, `isRandomized()` (which is used in isRequestComplete()) returns `true` and `fetchRandomnessAfter()` (which is used in randomNumber()) returns a valid random number as long as there is a successful request at or after the requested block number:

WitnetRandomnessV2.sol#L334-L336

```
/// @notice Returns `true` only if a successfull resolution from the Witnet
↪  blockchain is found for the first
```

```
/// @notice non-errored randomize request posted on or after the given block
↪  number.
function isRandomized(uint256 _blockNumber)
```

WitnetRandomnessV2.sol#L128-L136

```
/// @notice Retrieves the result of keccak256-hashing the given block number
↪  with the randomness value
/// @notice generated by the Witnet Oracle blockchain in response to the first
↪  non-errored randomize request solved
/// @notice after such block number.
/// @dev Reverts if:
/// @dev   i.   no `randomize()` was requested on neither the given block, nor
↪  afterwards.
/// @dev   ii.  the first non-errored `randomize()` request found on or after
↪  the given block is not solved yet.
/// @dev   iii. all `randomize()` requests that took place on or after the given
↪  block were solved with errors.
/// @param _blockNumber Block number from which the search will start
function fetchRandomnessAfter(uint256 _blockNumber)
```

For example, assume there are two requests:

- Request 1 made at `block.number = 100` failed.
- Request 2 made at `block.number = 101` was successful.

If `isRandomized()` and `fetchRandomnessAfter()` was called for block number 100, they would return `true` and a valid random number respectively. By extension, `RngWitnet.isRequestComplete()` would return `true` for request 1. However, `RngWitnet.isRequestFailed()` also returns `true` for request 1, forming a contradiction.

As such, it is possible for `RngWitnet.isRequestFailed()` to return `true` for a given `_requestId`, even when RngWitnet.isRequestComplete() returns `true` and RngWitnet.randomNumber() returns a valid random number.

In `DrawManager.startDraw()`, if `RngWitnet.isRequestFailed()` returns `true`, draw bots are allowed to call `startDraw()` again to submit a new request for the current draw to "retry" the randomness request:

DrawManager.sol#L238-L239

```
if (!rng.isRequestFailed(lastRequest.rngRequestId)) { // if the request failed
  revert AlreadyStartedDraw();
```

Since `isRequestFailed()` might wrongly return `true` as described above, it is possible for draw bots to retry a randomness request even when a random number

is actually available.

Additionally, it is possible for `startDraw()` to be called after a draw has concluded with `finishDraw()`. For example:

- `startDraw()` is called at `block.number = 100`.
- Witnet receives a randomness request from another user/protocol at `block.number = 101`.
- The randomness request at `block.number = 100` fails.
- The randomness request at `block.number = 101` succeeds.
- `finishDraw()` is called, which works as there is a successful request after `block.number = 100`.
- Now, if `startDraw()` is called, it will pass as `isRequestFailed()` returns `true` for the request at `block.number = 100`.

Note that it is not feasible for draw bots to check if `finishDraw()` has been called before calling `startDraw()`, as another draw bot can front-run their transaction calling `startDraw()` to call `finishDraw()`.

Impact

`startDraw()` can be called to retry the current draw even when (a) a random number is already available, or (b) when the current draw has already finished. (a) causes a loss of funds to other draw bots that called `startDraw()` in the current draw as their rewards are diluted, while (b) causes a loss of funds to the draw bots that call `startDraw()` after the draw has finished as they never receive rewards.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-rng-witnet/src/RngWitnet.sol#L115-L118

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-draw-manager/src/DrawManager.sol#L238-L239

Tool used

Manual Review

Recommendation

In `startDraw()`, consider checking `isRequestComplete()` to know if a random number is available:

```
-     if (!rng.isRequestFailed(lastRequest.rngRequestId)) { // if the request
↪  failed
+     if (rng.isRequestComplete(lastRequest.rngRequestId)) { // if the request
↪  failed
        revert AlreadyStartedDraw();
```

Discussion

10xhash

Escalate

Here we are speculating on the behavior of the bots which I believe would be out of scope. The bot would have to call the startDraw function even when the draw has finished or in an edge case (later rng request being reported earlier and earlier being reported ERROR) the bot would have the option to either call startDraw or call finishDraw

The recommendation is incorrect because it would allow all new requests to be overwritten. The current implementation is done with the assumption that witnet requests will be reported sequentially (which would be the case most of the time). In which case, as soon as the request results in an ERROR, the draw awarding can be reattempted. In case it doesn't happen sequentially and this request returns ERROR while the newer request returns a correct value, there is the option to either restart the draw or finish the draw and it would be depending on the bot's implementation which of these is performed

sherlock-admin3

> Escalate
>
> Here we are speculating on the behavior of the bots which I believe would be out of scope. The bot would have to call the startDraw function even when the draw has finished or in an edge case (later rng request being reported earlier and earlier being reported ERROR) the bot would have the option to either call startDraw or call finishDraw
>
> The recommendation is incorrect because it would allow all new requests to be overwritten. The current implementation is done with the assumption that witnet requests will be reported sequentially (which would be the case most of the time). In which case, as soon as the request results in an ERROR, the draw awarding can be reattempted. In case it doesn't happen sequentially and this request returns ERROR while the newer request returns a correct value, there is the option to either restart the draw or finish the draw and it would be depending on the bot's implementation which of these is performed

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

MiloTruck

> In case it doesn't happen sequentially and this request returns ERROR while the newer request returns a correct value, there is the option to either restart the draw or finish the draw and it would be depending on the bot's implementation which of these is performed

The current implementation allows for the following to occur:

- Bot calls `startDraw()` with the first request.
- First request returns ERROR.
- Second request returns a correct value.
- Bot calls `startDraw()` to restart the draw.
- Attacker front-runs the bot to call `finishDraw()`.
- Bot's call to `startDraw()` still executes and passes, causing it to have called `startDraw()` after `finishDraw()`.

There is no requirement for a bot to act abnormally here, it chooses the restart the draw as you have stated and ends up losing funds.

Even if the impacts/scenarios listed in my issue can only occur under an edge case or with bots behaving in a certain manner, I don't believe it is an acceptable risk for bots to potentially lose funds just by interacting with the protocol.

WangSecurity

To confirm:

> For example, assume there are two requests:

Request 1 made at block.number = 100 failed. Request 2 made at block.number = 101 was successful. If isRandomized() and fetchRandomnessAfter() was called for block number 100, they would return true and a valid random number respectively. By extension, RngWitnet.isRequestComplete() would return true for request 1. However, RngWitnet.isRequestFailed() also returns true for request 1, forming a contradiction.

Even if the request failed, `isRandomized`, `fetchRandomnessAfter` and `RngWitnet.isRequestComplete` would return true as if the request didn't fail. And `RngWitnet.isRequestFailed` would also return true, because it's not correctly

implemented (the root cause), could you please clarify this part briefly again, I feel like I'm missing something?

MiloTruck

@WangSecurity I'm not really sure what you don't understand.

`isRandomized()` checks if there is a successful request at a specific block or any block after it, while `isRequestFailed()` only checks if the request at a specific block failed. So in this scenario `isRandomized()` returns `true` for block 100 since a block after it contains a successful request (ie. request 2 at block 101), while `isRequestFailed()` returns `false` as request 1 at block 100 failed.

WangSecurity

Thank you for clarification, based on this and that, I believe it doesn't require for the bots to can in any strange way and it doesn't require any mistake on their end. Planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 10xhash: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/GenerationSoftware/pt-v5-rng-witnet/pull/8

10xhash

Fixed Now the exact request has to be solved for the query to be considered complete (instead of isRandomized returning true) disallowing a request to exist as both failed and complete at the same time

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-2: `drawTimeoutAt()` causes the prize pool to shut-down one draw earlier

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/28

Found by

MiloTruck, hash

Summary

The shutdown timestamp returned by `drawTimeoutAt()` is one draw period early, causing the protocol to shut down one draw earlier than expected.

Vulnerability Detail

In `PrizePool.sol`, the prize pool shuts down if the number of unawarded draws in a row is equal to `drawTimeout`:

PrizePool.sol#L283-L284

```
/// @notice The maximum number of draws that can be missed before the prize pool
 ↪  is considered inactive.
uint24 public immutable drawTimeout;
```

`drawTimeout` is used in `drawTimeoutAt()`, which determines the timestamp at which the pool shuts down:

PrizePool.sol#L973-L975

```
function drawTimeoutAt() public view returns (uint256) {
  return drawClosesAt(_lastAwardedDrawId + drawTimeout);
}
```

As seen from above, the pool shuts down at the close time of `drawId = _lastAwardedDrawId + drawTimeout`. However, this causes the pool to shut down one draw earlier than expected as draws can only be awarded after their close time in `awardDraw()`:

PrizePool.sol#L460-L465

```
uint24 awardingDrawId = getDrawIdToAward();
uint48 awardingDrawOpenedAt = drawOpensAt(awardingDrawId);
uint48 awardingDrawClosedAt = awardingDrawOpenedAt + drawPeriodSeconds;
```

```
if (block.timestamp < awardingDrawClosedAt) {
  revert AwardingDrawNotClosed(awardingDrawClosedAt);
}
```

To illustrate the problem:

- Assume the following:
  - `drawTimeout = 1`, which means the pool should shut down after one draw has been missed.
  - No draws have been awarded yet, so `_lastAwardedDrawId = 0`.
- The first draw to award is always `drawId = 1`.
- When `awardDraw()` is called before `drawId = 2`, the check in `awardDraw()` will revert as `block.timestamp` is less than the close time of `drawId = 1`.
- When `awardDraw()` is called during or after `drawId = 2` (ie. `block.timestamp` is greater than the close time of `drawId = 1`):
  - `_lastAwardedDrawId + drawTimeout = 0 + 1 = 1`
  - `drawTimeoutAt()` returns the close time of `drawId = 1`, so the pool has already shut down.
  - `awardDraw()` reverts due to the `notShutdown` modifier.

As seen from above, `awardDraw()` can never be called when `drawTimeout = 1`, even though a draw was never missed. This demonstrates how `drawTimeoutAt()` returns a timestamp one draw period early.

Impact

When `drawTimeout = 1`, the prize pool will never award prizes to any vaults and will immediately shut down, causing a loss of yield for depositors. Furthermore, this breaks core functionality of the protocol as the only incentive for users to deposit into vaults is the chance of winning a huge prize.

When `drawTimeout > 1`, the prize pool will shut down when `drawTimeout - 1` consecutive draws have been missed, which is one draw early.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L283-L284

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L973-L975

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L460-L465

Tool used

Manual Review

Recommendation

Modify `drawTimeoutAt()` to return the close time of `_lastAwardedDrawId + drawTimeout + 1`:

```
  function drawTimeoutAt() public view returns (uint256) {
-   return drawClosesAt(_lastAwardedDrawId + drawTimeout);
+   return drawClosesAt(_lastAwardedDrawId + drawTimeout + 1);
  }
```

Discussion

0xjuaan

Escalate

This issue should be low severity

The impact of this issue is that the prizePool will shutdown 1 draw earlier

This means that instead of shutting down 81 years later, the protocol will shutdown (81 years - 1 day) later.

Firstly the issue will only come into play around shutdown which is more than 81 years later. This amount of time is too large for the issue to be above low severity. For example, overflow when casting block.timestamp to uint32 has always been considered low, since the impact occurs in 82 years.

Secondly the fact that. the protocol shuts down 1 day earlier means that the protocol will shutdown 0.003% earlier. This is too small to be considered a medium severity issue.

sherlock-admin3

> Escalate
>
> This issue should be low severity
>
> The impact of this issue is that the prizePool will shutdown 1 draw earlier
>
> This means that instead of shutting down 81 years later, the protocol will shutdown (81 years - 1 day) later.

> Firstly the issue will only come into play around shutdown which is more than 81 years later. This amount of time is too large for the issue to be above low severity. For example, overflow when casting block.timestamp to uint32 has always been considered low, since the impact occurs in 82 years.
>
> Secondly the fact that. the protocol shuts down 1 day earlier means that the protocol will shutdown 0.003% earlier. This is too small to be considered a medium severity issue.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

MiloTruck

> This means that instead of shutting down 81 years later, the protocol will shutdown (81 years - 1 day) later.
>
> Firstly the issue will only come into play around shutdown which is more than 81 years later. This amount of time is too large for the issue to be above low severity. For example, overflow when casting block.timestamp to uint32 has always been considered low, since the impact occurs in 82 years.
>
> Secondly the fact that. the protocol shuts down 1 day earlier means that the protocol will shutdown 0.003% earlier. This is too small to be considered a medium severity issue.

The prize pool will also shutdown one draw earlier than intended when consecutive draws are not awarded. If `drawTimeout = 2`, the pool will shutdown after one draw has been missed, instead of two as intended. I've literally written this in my issue...

0xspearmint1

(The escalation was on my behalf)

The following is from the test suite

```
uint24 grandPrizePeriodDraws = 365;
uint48 drawPeriodSeconds = 1 days;
uint24 drawTimeout; // = grandPrizePeriodDraws * drawPeriodSeconds; // 1000 days;
```

The scenario of having 365 consecutive days without a single prize awarded is not realistic...

Even if this did happen, the protocol will shutdown in 364 days instead of 365. The

impact of shutting down 1 day earlier in such an extreme scenario is not medium severity.

Low severity is appropriate.

nevillehuang

Agree with @MiloTruck comments here. Even a single early shutdown would mean the prize pool will never award prizes to any vaults and will immediately shut down, leading to a loss of yield for depositors for that specific draw. Medium severity is appropriate

WangSecurity

Based on the Lead Judge's comment above, I believe Medium severity is appropriate here. Planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 0xjuaan: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-prize-pool/pull/117

10xhash

Fixed Documentation is updated to `draws that can pass` from `draws that can be missed` to match with the implementation and a minimum value of 2 draws is enforced

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-3: Price formula in `TpdaLiquidationPair._computePrice()` does not account for a jump in liquidatable balance

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/38

The protocol has acknowledged this issue.

Found by

0x73696d616f, 0xSpearmint1, MiloTruck

Summary

The linearly decreasing auction formula in `TpdaLiquidationPair._computePrice()` does not account for sudden increases in the vault's liquidatable balance, causing the price returned to be much lower than it should be.

Vulnerability Detail

The price paid by liquidation bots to liquidate the asset balance in a vault is calculated in `TpdaLiquidationPair._computePrice()` as shown:

TpdaLiquidationPair.sol#L191-L195

```
uint256 elapsedTime = block.timestamp - lastAuctionAt;
if (elapsedTime == 0) {
    return type(uint192).max;
}
uint192 price = uint192((targetAuctionPeriod * lastAuctionPrice) / elapsedTime);
```

As seen from above, the price paid decreases linearly over time from `targetAuctionPeriod * lastAuctionPrice` to 0. This allows the liquidation pair to naturally find a price that liquidation bots are willing to pay for the current liquidatable balance in the vault, which is calculated as shown:

TpdaLiquidationPair.sol#L184-L186

```
function _availableBalance() internal returns (uint256) {
    return ((1e18 - smoothingFactor) *
↪   source.liquidatableBalanceOf(address(_tokenOut))) / 1e18;
}
```

The vault's liquidatable balance is determined by `liquidatableBalanceOf()`:

PrizeVault.sol#L687-L709

```
function liquidatableBalanceOf(address _tokenOut) external view returns
↪  (uint256) {
    uint256 _totalDebt = totalDebt();
    uint256 _maxAmountOut;
    if (_tokenOut == address(this)) {
        // Liquidation of vault shares is capped to the mint limit.
        _maxAmountOut = _mintLimit(_totalDebt);
    } else if (_tokenOut == address(_asset)) {
        // Liquidation of yield assets is capped at the max yield vault withdraw
↪  plus any latent balance.
        _maxAmountOut = _maxYieldVaultWithdraw() +
↪  _asset.balanceOf(address(this));
    } else {
        return 0;
    }

    // The liquid yield is limited by the max that can be minted or withdrawn,
↪  depending on
    // `_tokenOut`.
    uint256 _availableYield = _availableYieldBalance(totalPreciseAssets(),
↪  _totalDebt);
    uint256 _liquidYield = _availableYield >= _maxAmountOut ? _maxAmountOut :
↪  _availableYield;

    // The final balance is computed by taking the liquid yield and multiplying
↪  it by
    // (1 - yieldFeePercentage), rounding down, to ensure that enough yield is
↪  left for
    // the yield fee.
    return _liquidYield.mulDiv(FEE_PRECISION - yieldFeePercentage,
↪  FEE_PRECISION);
}
```

Apart from the yield vault suddenly gaining yield, there are two reasons why the vault's liquidatable balance might suddenly increase:

- (1) The liquidatable balance is no longer restricted by the mint limit after someone withdraws from the vault. For example:
  - Assume `_tokenOut == address(this)`.
  - The vault is currently at the mint limit, so `_maxAmountOut = _mintLimit(_totalDebt) = 0`.
  - A user withdraws from the vault, causing the mint limit to decrease. Assume that `_maxAmountOut` is now `1000e18`, which means 1000 more

shares can be minted.

  - Since `_liquidYield` is restricted by `_maxAmountOut`, it jumps from `0` to `1000e18`, causing a sudden increase in the liquidatable balance.

- (2) The vault's owner calls `setYieldFeePercentage()` to decrease `yieldFeePercentage`. As seen from above, this will cause the returned value from `liquidatableBalanceOf()` to suddenly increase.

However, since the vault's liquidatable balance is not included in price calculation and the price decreases linearly, `TpdaLiquidationPair._computePrice()` cannot accurately account for instantaneous increases in `liquidatableBalanceOf()`.

Using (1) to illustrate this:

- Assume that the vault's token is USDT and the prize token is USDC.
- Since the vault is currently at its mint limit, the liquidatable balance is 0 USDT.
- Over time, the price calculated in `_computePrice()` decreases to near-zero to match the liquidatable balance.
- A user suddenly withdraws, which increases the mint limit and causes the liquidatable balance to jump to 1000 USDT.
- However, the price returned by `_computePrice()` will still remain close to 0 USDC.
- This allows a liquidation bot to swap 1000 USDT in the vault for 0 USDC.

As seen from above, when a sudden increase in liquidatable balance occurs, the price in `_computePrice()` could be too low due to the linearly decreasing auction.

Impact

When `liquidatableBalanceOf()` suddenly increases due to an increase in the mint limit or a reduction in `yieldFeePercentage`, liquidation bots can swap the liquidatable balance in the vault for less than its actual value. This causes a loss of funds for the vault, and by extension, all users in the vault.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-tpda-liquidator/src/TpdaLiquidationPair.sol#L191-L195

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-tpda-liquidator/src/TpdaLiquidationPair.sol#L184-L186

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-vault/src/PrizeVault.sol#L687-L709

Tool used

Manual Review

Recommendation

Consider using an alternative auction formula in `_computePrice()`, ideally one that takes into account the current liquidatable balance in the vault.

Discussion

sherlock-admin4

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> that is the reason of the MIN_PRICE state variable

trmid

If the yield source is expected to have large jumps in yield (by design or by the prize vault owner's actions) the prize vault owner can either:

1. set a suitable `smoothingFactor` on the liquidation pair to mitigate the effect within a reasonable efficiency target
2. pause liquidations during times of anticipated flux and set a new liquidation pair when ready (this would be good practice if suddenly lowering the yield fee percentage).

0xjuaan

Escalate

This issue should be informational

This is clearly a design decision by the protocol

The sponsor has also suggested some ways the prizeVault owner can mitigate this in the vault setup

Even if there is the odd upward fluctuation of yield the current design of the auction system ensures it will correct itself for future liquidations

sherlock-admin3

> Escalate
>
> This issue should be informational
>
> This is clearly a design decision by the protocol
>
> The sponsor has also suggested some ways the prizeVault owner can mitigate this in the vault setup
>
> Even if there is the odd upward fluctuation of yield the current design of the auction system ensures it will correct itself for future liquidations

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

MiloTruck

> This is clearly a design decision by the protocol
>
> The sponsor has also suggested some ways the prizeVault owner can mitigate this in the vault setup

I don't see how the auction mechanism being a design decision invalidates this issue. I've clearly pointed out how this design decision is incapable of handling upward yield fluctuations, which is an actual problem.

The sponsor has pointed out ways which, in my opinion, only *partially* mitigate the problem presented. The owner has to either has to temporarily stop liquidations or reduce the percentage of yield that can be liquidated to smooth out the upward fluctuation. Note that both ways are a form of owner intervention, and I don't believe it is documented anywhere on how the owner should deal with upward yield fluctuations.

> Even if there is the odd upward fluctuation of yield the current design of the auction system ensures it will correct itself for future liquidations

The prize vault still loses funds for the current liquidation, so this doesn't invalidate anything.

nevillehuang

Agree with @MiloTruck comments.

WangSecurity

As the report says, the issue may cause loss of funds to users, hence, I believe design decision rule is not appropriate here. Planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 0xjuaan: rejected

Issue M-4: `TpdaLiquidationPair.swapExactAmountOut()` can be DOSed by a vault's mint limit

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/39

The protocol has acknowledged this issue.

Found by

0xSpearmint1, MiloTruck

Summary

By repeatedly DOSing `TpdaLiquidationPair.swapExactAmountOut()` for a period of time, an attacker can swap the liquidatable balancce in a vault for profit.

Vulnerability Detail

When liquidation bots call `TpdaLiquidationPair.swapExactAmountOut()`, they specify the amount of tokens they wish to receive in `_amountOut`. `_amountOut` is then checked against the available balance to swap from the vault:

TpdaLiquidationPair.sol#L141-L144

```
uint256 availableOut = _availableBalance();
if (_amountOut > availableOut) {
    revert InsufficientBalance(_amountOut, availableOut);
}
```

The available balance to swap is determined by the liquidatable balance of the vault: TpdaLiquidationPair.sol#L184-L186

```
function _availableBalance() internal returns (uint256) {
    return ((1e18 - smoothingFactor) *
↪   source.liquidatableBalanceOf(address(_tokenOut))) / 1e18;
}
```

However, when the output token from the swap (ie. `tokenOut`) is vault shares, `PrizeVault.liquidatableBalanceOf()` is restricted by the mint limit:

PrizeVault.sol#L687-L709

```
function liquidatableBalanceOf(address _tokenOut) external view returns
↪   (uint256) {
    uint256 _totalDebt = totalDebt();
```

```
    uint256 _maxAmountOut;
    if (_tokenOut == address(this)) {
        // Liquidation of vault shares is capped to the mint limit.
        _maxAmountOut = _mintLimit(_totalDebt);
    } else if (_tokenOut == address(_asset)) {
        // Liquidation of yield assets is capped at the max yield vault withdraw
↪   plus any latent balance.
        _maxAmountOut = _maxYieldVaultWithdraw() +
↪   _asset.balanceOf(address(this));
    } else {
        return 0;
    }

    // The liquid yield is limited by the max that can be minted or withdrawn,
↪   depending on
    // `_tokenOut`.
    uint256 _availableYield = _availableYieldBalance(totalPreciseAssets(),
↪   _totalDebt);
    uint256 _liquidYield = _availableYield >= _maxAmountOut ? _maxAmountOut :
↪   _availableYield;

    // The final balance is computed by taking the liquid yield and multiplying
↪   it by
    // (1 - yieldFeePercentage), rounding down, to ensure that enough yield is
↪   left for
    // the yield fee.
    return _liquidYield.mulDiv(FEE_PRECISION - yieldFeePercentage,
↪   FEE_PRECISION);
}
```

This means that if the amount of shares minted is close to `type(uint96).max`, the available balance in the vault (ie. `_liquidYield`) will be restricted by the remaining number of shares that can be minted.

However, an attacker can take advantage of this to force all calls to `TpdaLiquidationPair.swapExactAmountOut()` to revert:

- Assume a vault has the following state:
  - The amount of `_availableYield` in the vault is `1000e18`.
  - The amount of shares currently minted is `type(uint96).max - 2000e18`, so `_liquidYield` is not restricted by the mint limit.
  - `yieldFeePercentage = 0` and `smoothingFactor = 0`.
- A liquidation bot calls `swapExactAmountOut()` with `_amountOut = 1000e18`.
- An attacker front-runs the liquidation bot's transaction and deposits `1000e18 +`

`1` tokens, which mints the same amount of shares:

  - The amount of shares minted is now `type(uint96).max - 1000e18 + 1`, which means the mint limit is `1000e18 - 1`.
  - As such, the available balance in the vault is reduced to `1000e18 - 1`.

- The liquidation bot's transaction is now executed:
  - In `swapExactAmountOut()`, `_amountOut > availableOut` so the call reverts.

Note that the `type(uint96).max` mint limit is reachable for tokens with low value. For example, PEPE has 18 decimals and a current price of $0.00001468, so `type(uint96).max` is equal to $1,163,070 worth of PEPE. For tokens with a higher value, the attacker can borrow funds in the front-run transaction, and back-run the victim's transaction to return the funds.

This is an issue as the price paid by liquidation bots for the liquidatable balance decreases linearly over time:

TpdaLiquidationPair.sol#L191-L195

```
uint256 elapsedTime = block.timestamp - lastAuctionAt;
if (elapsedTime == 0) {
    return type(uint192).max;
}
uint192 price = uint192((targetAuctionPeriod * lastAuctionPrice) / elapsedTime);
```

As such, an attacker can repeatedly perform this attack (or deposit sufficient funds until the mint limit is 0) to prevent any liquidation bot from swapping the liquidatable balance. After the price has decreased sufficiently, the attacker can then swap the liquidatable balance for himself at a profit.

Impact

By depositing funds into a vault to reach the mint limit, an attacker can DOS all calls to `TpdaLiquidationPair.swapExactAmountOut()` and prevent liquidation bots from swapping the vault's liquidatable balance. This allows the attacker to purchase the liquidatable balance at a discount, which causes a loss of funds for users in the vault.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-tpda-liquidator/src/TpdaLiquidationPair.sol#L141-L144

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-tpda-liquidator/src/TpdaLiquidationPair.sol#L184-L186

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-vault/src/PrizeVault.sol#L687-L709

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-tpda-liquidator/src/TpdaLiquidationPair.sol#L191-L195

Tool used

Manual Review

Recommendation

Consider implementing `liquidatableBalanceOf()` and/or `_availableBalance()` such that it is not restricted by the vault's mint limit.

For example, consider adding a `_tokenOut` parameter to `TpdaLiquidationPair.swapExactAmountOut()` for callers to specify the output token. This would allow liquidation bots to swap for the vault's asset token, which is not restricted by the mint limit, when the mint limit is reached.

Discussion

trmid

If the prize vault owner is deploying with an asset that is expected to be close to the TWAB limit in deposits, they can set a liquidation pair that liquidates the yield as assets instead of shares to bypass this limit.

However, it would not be advisable to deploy a prize vault with an asset that is expected to reach the clearly documented TWAB limits.

nevillehuang

@WangSecurity @trmid I overlooked this issue. This should be a duplicate of #19, still considering validity and severity

Hash01011122

@nevillehuang This appears to be dup of #88 not #19

MiloTruck

> This appears to be dup of #88 not #19

It's not a dupe of #88, #88 is describing how the calculation in `liquidatableBalanceOf()` doesn't take into account the mint limit when `_tokenOut == address(_asset)`. I intentionally didn't submit it since I believe it's low severity.

This issue describes how you can intentionally hit the mint limit to prevent liquidations from occurring.

MiloTruck

> I overlooked this issue. This should be a duplicate of #19, still considering validity and severity

@nevillehuang Would just like to highlight that the main bug here is being able to DOS liquidations due to the mint limit, being able to reduce the auction price is just one of the impacts this could have. This is not to be confused with #38, which doesn't need liquidations to be DOSed to occur.

Perhaps this issue is more similar to #22 than #19.

0x73696d616f

This is not a duplicate of #88, it's a duplicate of #19. I also did not intentionally submit this issue because it is a design choice.

This one, #19 and all the dupes that depend on reaching the mint limit and bots not being able to liquidate because an attacker is DoSing them or similar are at most medium because:

1. The owner can set liquidations to be based on the asset out.
2. If any user withdraws, bots can liquidate anyway.
3. Bots can use flashbots.
4. For these issues to be considered, a large % of the pool is required by the attacker.
5. DoS requiring the attacker to keep DoSing are considered 1 block DoSes.

From these points, we can see that it is impossible to keep the DoS for a long period, so it is either considered a design choice or medium.

0xspearmint1

1. Even if the owner sets liquidations to be based on asset out the `_enforceMintLimit` function is still called and will revert the liquidation, as long as yieldFee is > 0 which it will be.

```
if (_tokenOut == address(_asset)) {
            _enforceMintLimit(_totalDebtBefore, _yieldFee);
```


WangSecurity

I agree this should be a duplicate of #19 (even though it may be more similar to #22, #22 is a duplicate of #19 itself). Since there is no escalation, planning to just duplicate it.

WangSecurity

Based on the discussion under #19, this report will be the main of the new family. Medium severity, because the attacker has to constantly keep the mint limit reached, which even with small tokens (SHIB or LADYS) needs large capital (flash loans cannot be used).

Issue M-5: The claimer's fee will be stolen by the winner

Source:
https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/73

Found by

0x73696d616f, cu5t0mPe0, jovi, ydlee

Summary

The user calls `claimPrizes` to collect prizes for the winner, earning some rewards in the process. However, during this process, the winner can steal the fees collected by the user without paying any gas fees.

Vulnerability Detail

The user calls `claimPrizes` to collect prizes for the winner, thereby earning a reward fee.

Throughout the process, the winner can set hooks before and after calling `prizePool.claimPrize`. The winner can set an `afterClaimPrize` hook and then call `claimer.claimPrizes` again within this `afterClaimPrize` hook, thereby earning reward fees without using any gas. As a result, the user can only collect the reward fee for one winner, while the remaining reward fees for other winners will all be collected by the first winner.

Consider the following scenario:

1. User A calls `claimer.claimPrizes` to claim prizes, setting the winners as [B, C, D, E, F] (B, C, D, E, F are all winners).
2. User B notices in the mempool that User A is about to call `claimer.claimPrizes` to claim prizes for others. User B then calls `setHooks` to set the `afterClaimPrize` function, which implements a logic to loop call `claimer.claimPrizes` with winners set as [C, D, E, F].
3. User A's transaction is executed, but since User B has already claimed the rewards for C, D, E, and F, User A's attempts to claim prizes for C, D, E, and F will revert. However, due to the `try-catch`, User A's transaction will continue executing. In the end, User A will only receive the reward fee for User B.

This is essentially the same attack method as the previous PoolTogether vulnerability. The last audit did not completely fix it. The difference this time is that the hook uses `afterClaimPrize`.

https://code4rena.com/reports/2024-03-pooltogether#m-01-the-winner-can-steal-claimer-fees-and-force-him-to-pay-for-the-gas

Impact

The user will lose the reward fee

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-vault/src/abstract/Claimable.sol#L110-L117

Tool used

Manual Review

Recommendation

If the hook's purpose is for managing whitelists and blacklists, PoolTogether can create a template contract that users can manage. This way, the hook can only be set to the template contract and cannot perform additional operations.

If users are allowed to perform any operation, there is no good solution to this problem. The only options are to limit the gas, use reentrancy locks, or remove the after hook altogether.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> if the attacker can front-run one prize then he is better off front-running the whole claimPrizes call and earn even more reward

trmid

The risks of front-running claims is a known issue from previous audits: https://code4rena.com/reports/2023-07-pooltogether#m-24-claimerclaimprizes-can-be-front-runned-in-order-to-make-losses-for-the-claim-bot

Although the method described here is slightly more complex, the end result is still very similar to the more straightforward approach of frontrunning the entire tx.

It is also worth noting that the prize hooks have a gas limit of 150k gas, which would cause this strategy to fail when trying to frontrun a large number of prizes.

nevillehuang

I believe medium severity is appropriate here, this issue wasn't highlighted as a known issue/risk, and watsons has highlighted another vector that allows stealing of claimer fees

InfectedIsm

Escalate

The reason I think it shouldn't be valid is because its a front-run, and any profitable action can be front-run and nothing can be done against this. We can also show that a simple copy-paste front-run of the tx is always a better solution than that one.

Here the attacker need (1) to detect the user A call (2) front-run it to update its hooks

But if the user is able to do do this, then why wouldn't it simply copy-paste the whole user A call, and put his own claiming call before user A? He would get even more reward and more disturbance.

Following the described attack path of this submission, the user will only be able to claim 2 prizes (before and after hooks), which will be less beneficial than front-running the whole claiming array with a standard claim call.

There is no way to defend against a front-run unless using the proposed remediation (using whitelist/blacklist), but this totally goes against the way the protocol is designed as a permissionnes protocol

sherlock-admin3

> Escalate
>
> The reason I think it shouldn't be valid is because its a front-run, and any profitable action can be front-run and nothing can be done against this. We can also show that a simple copy-paste front-run of the tx is always a better solution than that one.
>
> Here the attacker need (1) to detect the user A call (2) front-run it to update its hooks
>
> But if the user is able to do do this, then why wouldn't it simply copy-paste the whole user A call, and put his own claiming call before user A? He would get even more reward and more disturbance.
>
> Following the described attack path of this submission, the user will only be able to claim 2 prizes (before and after hooks), which will be less beneficial than front-running the whole claiming array with a standard claim call.

> There is no way to defend against a front-run unless using the proposed remediation (using whitelist/blacklist), but this totally goes against the way the protocol is designed as a permissionnes protocol

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0jovi0

> Escalate
>
> The reason I think it shouldn't be valid is because its a front-run, and any profitable action can be front-run and nothing can be done against this. We can also show that a simple copy-paste front-run of the tx is always a better solution than that one.
>
> Here the attacker need (1) to detect the user A call (2) front-run it to update its hooks
>
> But if the user is able to do do this, then why wouldn't it simply copy-paste the whole user A call, and put his own claiming call before user A? He would get even more reward and more disturbance.
>
> Following the described attack path of this submission, the user will only be able to claim 2 prizes (before and after hooks), which will be less beneficial than front-running the whole claiming array with a standard claim call.
>
> There is no way to defend against a front-run unless using the proposed remediation (using whitelist/blacklist), but this totally goes against the way the protocol is designed as a permissionnes protocol

The other dupes are not about the front-run but about the hooks allowing gas or prize fees to be stolen. Issue #82 and #161 describe a beforeClaimPrize hook that effectively steals prize fees without any mempool manipulation whatsoever.

0x73696d616f

It is not true that it would always be better to frontrun the tx and execute the same transaction because there is a bigger upfront cost and there is the risk that some claims have been made and a lot of gas is wasted. This allows gaming the claiming mechanism and could be fixed by simply adding a reentrancy lock. The bot would take a loss from this. The bot gives `300k` free gas to the user per prize claimed. As the bots will collect a lot of prizes for each user, the gas given to users is `300k*n`, where `n` is the number of prizes claimed. Thus, a simple frontrun to change the

logic of a hook, will cost something like 30k gas, gives the user `300k * n` gas to claim prizes, at no risk, which is obviously problematic.

0x73696d616f

Also, is it possible to trick the simulation of the transaction so no frontrunning is needed? I am really curious about this. I was thinking about using `block.timestamp`, but there is a chance that the simulation fails.

WangSecurity

I agree with the comments above, the fact that there may be a "better" attack path (which is explained not to be better in the comment above) doesn't mean this attack path won't be taken or it's negligible. Moreover, if the mitigation is not the best, it doesn't mean the issue is invalid.

Planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- InfectedIsm: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-claimer/pull/32

10xhash

Fixed. Now re-entrancy guard is added to to the Claimer contract which disallows claiming prizes by reentering through the same claimer contract. If in future vaults are launched with a different claimer contract and bots attempt to claim prizes for both these vaults in the same tx, then the same issue would arise again

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-6: `PUSH0` opcode Is Not Supported on Linea yet

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/79

The protocol has acknowledged this issue.

Found by

0xSpearmint1, aman, elhaj, volodya

Summary

Vulnerability Detail

- The current codebase is compiled with Solidity `version 0.8.24`, which includes the `PUSH0` opcode in the compiled bytecode. According to the README, the protocol will be deployed on the Linea network.
- This presents an issue because Linea does not yet support the `PUSH0` opcode, which can lead to unexpected behavior or outright failures when deploying and running the smart contracts.see here

Impact

- Deploying the protocol on `Linea` with the current Solidity version (0.8.24) may result in unexpected behavior or failure due to the unsupported `PUSH0` opcode.

Code Snippet

- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L2C1-L3C1

Tool used

Manual Review

Recommendation

- for Linea you may consider to use version 0.8.19 to compile .

Discussion

sherlock-admin3

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> contract will simply not deploy so no risk/loss__ see bullet 24 of invalid findings in sherlock rules

nevillehuang

Valid medium, since if current contract is deployed as is, it will have issues and as per noted in contest details, so the contest details will override sherlock rule 24 since it is stated as follows:

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.

10xhash

Escalate

push0 is not present in the generated bytecode of any contract

sherlock-admin3

> Escalate
>
> push0 is not present in the generated bytecode of any contract

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

elhajin

Foundry has a default **evm_version** set to **Paris**. If you compile directly with this default setting, you won't get the `PUSH0` opcode in the bytecode. However, by setting the environment variable **FOUNDRY_EVM_VERSION** to **Shanghai** or **Cancun**, or by compiling with the flag `--evm-version shanghai`, for example, the `PUSH0` opcode will be introduced.

We don't know how devs will compile the codebase . Given this statement from the README:

> We're interested to know if there will be any issues deploying the code as-is to any of these chains, and whether their opcodes fully support our application.

I believe this is at least a medium severity issue according to Sherlock's rules.

InfectedIsm

Sherlock's rules :

> V. How to identify a medium issue:
>
> - Causes a loss of funds but requires certain external conditions or specific states, or a loss is highly constrained. The losses must exceed small, finite amount of funds, and any amount relevant based on the precision or significance of the loss.
> - Breaks core contract functionality, rendering the contract useless or leading to loss of funds.

But here there are no loss of funds, neither core contract functionality broken/contract useless as the contract will simply not be deployable if I'm not mistaken

But not only that, this is considered invalid by Sherlock's rules:

> VII. List of Issue categories that are not considered valid: ... 24. Using Solidity versions that support EVM opcodes that don't work on networks on which the protocol is deployed is not a valid issue beacause one can manage compilation flags to compile for past EVM versions on newer Solidity versions.

elhajin

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. **Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity**. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.

nevillehuang

Agree with comment here and here, by sherlocks hierarchy of truth, escalations should be rejected:

> If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules.

WangSecurity

Firstly, want to clarify that the hierarchy of truth quoted in the couple of messages above is the new one, while this contest uses an old version of rules. See here. Now, each contest has its rule version (the current rules at the contest start) and you can see the link on the contest's page (below the end date).

Secondly, as I understand the current code can indeed be deployed on Linea, but using the older compiler version, correct? Taking the fact, that we don't know how the contracts will be compiled, into consideration, I believe the Rule 24 about EVM Opcodes still applies here and the issue is low severity. Unless this paragraph is not wrong, planning to accept the escalation and invalidate the issue.

nevillehuang

@WangSecurity

the old rules also point to read me overriding sherlock rules. Also when is the new rules introduced and from which contest is it applied (should make an announcement for important changes like this)

> In case of conflict between information in the README vs Sherlock rules, the README overrides Sherlock rules

the contracts will compile correctly because of the foundry configurations mentioned here. But with the given solidity version, push0 will cause deployment reverts given linea does not support this opcode.

It is very likely there will be a zero constant somewhere within the codebase pushed onto the stack, but I agree watsons should verify this.

elhajin

Sorry didn't know about which rules are used.. but I think my point still hold

> SHERLOCK RULES (for this contest): In case of conflict between information in the README vs Sherlock rules, the README overrides Sherlock rules
>
> CONTEST README : We're interested to know if there will be any issues deploying the code as-is to any of these chains, and whether their **opcodes fully support** our application

- the compiling of the code depends on the command you run .. and how you set your environment variables and **we don't know** how Devs will compile the code

amankakar

> CONTEST README : We're interested to know if there will be any issues deploying the code as-is to any of these chains, and whether their opcodes fully support our application

The contest read me has a high priority over the Sherlock rules and the team is really interested to know if there is any opcode incompatibility, This finding provides the clear explanation of issue which will result in DoS if deployed on Linea. Hence the sponsor confirmed it which means they did not know about it.

WangSecurity

Excuse me for the confusion about the rules, I just meant to remind you about the commits each contest has and didn't mean to say that it somehow changes the situation.

I believe this issue is indeed medium severity cause the protocol asked about issues with the contracts as-is and their opcodes. Planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 10xhash: rejected

Issue M-7: Potential ETH Loss Due to transfer Usage in Requestor Contract on zkSync

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/80

Found by

0x73696d616f, 0xAadi, MiloTruck, elhaj, trachev

Summary

- The Requestor contract uses `transfer` to send `ETH` which has the risk that it will not work if the gas cost increases/decrease(low Likelihood), but it is highly likely to fail on `zkSync` due to gas limits. This may make users' `ETH` irretrievable.

Vulnerability Detail

- Users (or bots) interact with RngWitnet to request a random number and start a draw in the `DrawManager` contract. To generate a random number, users must provide some `ETH` that will be sent to WitnetRandomness to generate the random number.

```
function startDraw(uint256 rngPaymentAmount, DrawManager _drawManager, address
↪  _rewardRecipient) external payable returns (uint24) {
        (uint32 requestId,,) = requestRandomNumber(rngPaymentAmount);
        return _drawManager.startDraw(_rewardRecipient, requestId);
}
```

- The `ETH` sent with the transaction may or may not be used (if there is already a request in the same block, it won't be used). Any remaining or unused `ETH` will be sent to Requestor, so the user can withdraw it later.
- The issue is that the `withdraw` function in the `Requestor` contract uses `transfer` to send `ETH` to the receiver. This may lead to users being unable to withdraw their funds .

```
function withdraw(address payable _to) external onlyCreator returns (uint256) {
       uint256 balance = address(this).balance;
>>       _to.transfer(balance);
       return balance;
}
```

- The protocol will be deployed on different chains including `zkSync`, on `zkSync` the use of `transfer` can lead to issues, as seen with

921 ETH Stuck in zkSync Era.since it has a fixed amount of gas '23000' which won't be anough in some cases even to send eth to an `EOA`, It is explicitly mentioned in their docs to not use the `transfer` method to send `ETH` here.

notice that in case `msg.sender` is a contract that have some logic on it's receive or fallback function the `ETH` is definitely not retrievable. since this contract can only withdraw eth to it's own addres which will always revert.

Impact

- Draw Bots' `ETH` may be irretrievable or undelivered, especially on zkSync, due to the use of `.transfer`.

Code Snippet

- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-rng-witnet/src/Requestor.sol#L27-L30

Tool used

Manual Review , Foundry Testing

Recommendation

- recommendation to use `.call()` for `ETH` transfer.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

invalid__ only affect bots that deliberately chose to set code inside the fallback function

trmid

The zkSync doc link provided in the issue does not seem to have information on this behaviour anymore. Is there an updated link on the issue?

nevillehuang

request poc

sherlock-admin4

PoC requested from @elhajin

Requests remaining: **1**

elhajin

- here the correct quote from the docs : Use call over .send or .transfer

InfectedIsm

Escalate

This submission make the assumption that the caller will set code in their callback to process received ETH. This pose the assumption that those callers will not be aware of the limitation of the transfer/send behavior on these chains. Setting code in the callback isn't required at all for the caller. By simply not setting code in their callback, the issue disappear, thus this is a user error and not a vulnerability of the codebase, but rather a proposition of improvement of the implemented mechanism.

from the doc: https://docs.zksync.io/build/developer-reference/best-practices#use-call-over-send-or-transfer

> Avoid using payable(addr).send(x)/payable(addr).transfer(x) because the 2300 gas stipend may not be enough for such calls, especially if it involves state changes that require a large amount of L2 gas for data. Instead, we recommend using call.

sherlock-admin3

> Escalate
>
> This submission make the assumption that the caller will set code in their callback to process received ETH. This pose the assumption that those callers will not be aware of the limitation of the transfer/send behavior on these chains. Setting code in the callback isn't required at all for the caller. By simply not setting code in their callback, the issue disappear, thus this is a user error and not a vulnerability of the codebase, but rather a proposition of improvement of the implemented mechanism.
>
> from the doc: https://docs.zksync.io/build/developer-reference/best-practices#use-call-over-send-or-transfer
>
> > Avoid using payable(addr).send(x)/payable(addr).transfer(x) because the 2300 gas stipend may not be enough for such calls, especially if it involves state changes that require a large amount of L2 gas for data. Instead, we recommend using call.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

I wouldn't call this user error because any address can be used to transfer funds to creator. Also it does not just affect caller with no code and there is clear evidence from the sponsor acknowledgement and lack of details in contest that the sponsor wasn't aware of this issue.

0xAadi

Disagreeing with the Escalation

> By simply not setting code in their callback, the issue disappears.

Removing the callbacks from the caller will not solve this issue.

The core issue is not related to the use of code in the callback; rather, it is a limitation of the ZkSync chain. The core issue arises due to the dynamic gas measurement followed in ZkSync.

ZkSync usually requires more than 2300 gas to process `transfer()`/`send()` functions. Therefore, the `withdraw()` function will always revert due to the lack of enough gas to process `transfer()` on ZkSync.

ZkSync explicitly mentions this in their security and best practice documentation:

> Avoid using payable(addr).send(x)/payable(addr).transfer(x) because the 2300 gas stipend may not be enough for such calls.

The situation worsens if there are any callback functionalities in the caller.

This is a known issue in ZkSync, as evidenced by previous occurrences in other protocols:

- 921 ETH Stuck in zkSync Era
- ZkSync Twitter Post

Note: Please see issues #24 and #139, which explain the issue in detail.

InfectedIsm

Fair enough, I wasn't able to find that even without callback execution it could spend more than 2300. Sorry for the misunderstanding, and thanks for the detailed explanation!

0xAadi

In addition to my previous comment:

This issue causes a loss of ETH in ZkSync.

According to Sherlock's docs, this issue should be considered as valid HIGH:

> IV. How to identify a high issue:
>
> 1. Definite loss of funds without (extensive) limitations of external conditions.

Please consider upgrading the severity to HIGH.

Please find my thoughts on the duplicates below:

#24 and #139 both identify the same issue in ZkSync. #94 missed the aforementioned issue but did identify another medium-risk attack path. #119 missed both of the aforementioned attack paths but identified a more general issue.

WangSecurity

Since the escalator agrees with the other side (correct me if it's wrong) and based on the discussion above, I believe this issue should remain as it is. Planning to reject the escalation.

0xAadi

> the escalator agrees with the other side (correct me if it's wrong)

true, reference

@WangSecurity, As I mentioned in this duplicate and here, This vulnerability can cause loss of ETH on the zkSync chain.

Therefore, please consider upgrading the severity of this issue from medium to high.

WangSecurity

Is there a specific number of ETH that has to be transferred in that case, or it can be as small as 1 wei? And as I understand there's still a possibility the `transfer` would work correctly?

InfectedIsm

Rng draw rewards are very low (<0.0001 WETH), as it can be seen on the Pool Together dashboard: https://analytics.cabana.fi/optimism The loss that will be experienced by rng drawer isn't large enough to be an argument for a high severity imo

WangSecurity

I agree with the comment above, I believe it's a small value and only caused on one chain, hence, I believe medium is more appropriate.

The decision remains the same, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- InfectedIsm: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-rng-witnet/pull/9

10xhash

Fixed. Now .call is used instead of .transfer

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-8: Claimers Cannot Claim Prizes When Last Tier Liquidity is 0, Preventing Winners from Receiving Their Prizes

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/84

Found by

elhaj

Summary

- The `Claimer` contract's `claimPrizes` function can revert when the prize for the tier `(numberOfTiers - 3)` is 0. This prevents all winners in this draw from receiving their prizes (through claimer) and stops honest claimers from claiming rewards due to the transaction reverting.

Vulnerability Detail

- The PrizePool contract holds the prizes and ensures that vault users who contributed to the `PrizePool` have the chance to win prizes proportional to their contribution and twab balance in every draw. On the other hand, the Claimer contract facilitates the claiming of prizes on behalf of winners so that winners automatically receive their prizes. **An honest claimer should always have the ability to claim prizes for correct winners in every draw.**
- The ClaimPrizes function in the Claimer contract allows anyone to call it to claim prizes for winners. The caller (bots) computes the winners off-chain and they are incentivized through an auction mechanism where the reward increases over time if prizes are not claimed promptly.

```
    function claimPrizes(IClaimable _vault, uint8 _tier, address[] calldata
↪   _winners, uint32[][] calldata _prizeIndices, address _feeRecipient,
↪   uint256 _minFeePerClaim)
        external
        returns (uint256 totalFees)
    {
    //some code ...

        /**
         * If the claimer hasn't specified both a min fee and a fee
↪   recipient, we assume that they don't
         * expect a fee and save them some gas on the calculation.
```

```
         */
        if (!feeRecipientZeroAddress) {
>>        feePerClaim = SafeCast.toUint96(_computeFeePerClaim(_tier,
↪  _countClaims(_winners, _prizeIndices), prizePool.claimCount()));
            if (feePerClaim < _minFeePerClaim) {
                revert VrgdaClaimFeeBelowMin(_minFeePerClaim, feePerClaim);
            }
        }

        return feePerClaim * _claim(_vault, _tier, _winners, _prizeIndices,
↪  _feeRecipient, feePerClaim);
    }
```

- Notice that the function calls the internal function `_computeFeePerClaim()` to compute the fee the claimer bot will receive based on the auction condition at this time. In `_computeFeePerClaim()`, another internal function `_computeDecayConstant` is called to compute the decay constant for the `Variable Rate Gradual Dutch Auction`:

```
    function _computeFeePerClaim(uint8 _tier, uint256 _claimCount, uint256
↪  _claimedCount) internal view returns (uint256) {
    //some code ..
>>    SD59x18 decayConstant = _computeDecayConstant(targetFee, numberOfTiers);
    // more code ...
    }
    function _computeDecayConstant(uint256 _targetFee, uint8 _numberOfTiers)
↪  internal view returns (SD59x18) {
>>    uint256 maximumFee = prizePool.getTierPrizeSize(_numberOfTiers - 3);
>>    return LinearVRGDALib.getDecayConstant(LinearVRGDALib.getMaximumPriceDelta
↪  Scale(_targetFee, maximumFee, timeToReachMaxFee));
    }
```

- In `_computeDecayConstant`, the `maximumFee` is obtained as the prize of the `tier(_numberOfTiers - 3)`, which is the last tier before canary tiers. This value is then fed into the `LinearVRGDALib.getMaximumPriceDeltaScale()` function.
- The issue arises when the `maximumFee` is zero, which can happen if there is no liquidity in this tier. With `maximumFee = 0`, the internal call to `LinearVRGDALib.getMaximumPriceDeltaScale()` will revert specifically in `wadLn` function when input is `0`, causing the transaction to revert:

```
    function getMaximumPriceDeltaScale(uint256 _minFee, uint256 _maxFee, uint256
↪  _time) internal pure returns (UD2x18) {
        return ud2x18(SafeCast.toUint64(uint256(wadExp(wadDiv(
            wadLn(wadDiv(SafeCast.toInt256(_maxFee),
↪  SafeCast.toInt256(_minFee))),//@audit : this will be zero
```

```
            SafeCast.toInt256(_time)) / 1e18)));
    }

    function wadLn(int256 x) pure returns (int256 r) {
    unchecked {
>>    require(x > 0, "UNDEFINED");
        // ... more code ..
    }
    }
```

- This means that even if a winner wins, the claimer won't be able to claim the prize for them if `tier(_numberOfTiers - 3).prize = 0`, unless they set fees to 0. However, no claimer will do this since they would be paying gas for another user's prize. This will lead to winners not receiving their prizes even when they have won, and it prevents the honest claimer from performing its expected job due to external conditions.
- The winner in this case may win prizes from tiers less than `_numberOfTiers - 3`, which are rarer and higher prizes (they can't win prizes from tier `(_numberOfTiers - 3)` since it has 0 liquidity).
- The `tier(_numberOfTiers - 3).prize` can be 0 in different scenarios (less liquidity that round to zero when devid by prize count,high `tierLiquidityUtilizationRate` ..ect). Here is a detailed example explain one of the scenarios that can lead to `tier(_numberOfTiers - 3).prize= 0` :
- ***Example Scenario***

1. **Initial Setup**:
    - Assume we have 4 tiers `[0, 1, 2, 3]`.
    - Current draw is 5.
    - `rewardPerToken = 2`.
2. **Awarding Draw 5**:
    - `rewardPerToken` becomes 3.
    - Prizes are claimed, and all canary prizes are claimed due to a lot of liquidity contribution in draw 5.
    - Liquidity for each tier after draw 5: | Tier | Reward Per Token (rpt) | |------|----------------------| | t0 | 0 | | t1 | 0 | | t2 | 3 | | t3 | 3 |
    - Notice that the remaining liquidity in tiers 2 and 3 (canary tiers) is 0 now.
3. **Awarding Draw 6**:
    - Time passes, and the next draw is 6.

- Draw 6 is awarded, but there were no contributions.
- Since the claim count was high in the previous draw, the number of tiers is incremented to 5.
- Reclaim liquidity in both previous tiers 2 and 3, but there is no liquidity to reclaim.
- There is no new liquidity to distribute, so the `rewardPerToken` remains 3.
- Liquidity for each tier in draw 6: | Tier | Reward Per Token (rpt) | |------|----------------------| | t0 | 0 | | t1 | 0 | | t2 | 3 | | t3 | 3 | | t4 | 3 |

4. **Claiming Prizes**:

- A claimer computes the winners (consuming some resources) and calls `claimPrizes` with the winners.
- The `uint256 maximumFee = prizePool.getTierPrizeSize(_numberOfTiers - 3);` (tier2) will be 0 since there was no liquidity .
- This causes `wadLn(0)` to revert when trying to compute the claimer fees thus tx revert and claimer can't claim any prize.

Impact

- claimers cannot claim prizes for legitimate winners due to the transaction reverting.
- all Users who won the prizes in this awarded draw won't receive their prizes (unless they claim for them selfs with 0 fees which unlikely).

Code Snippet

- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-claimer/src/Claimer.sol#L221
- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-claimer/src/Claimer.sol#L272-L279
- https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-claimer/src/libraries/LinearVRGDALib.sol#L114
- https://github.com/transmissions11/solmate/blob/c892309933b25c03d32b1b0d674df7ae292ba925/src/utils/SignedWadMath.sol#L165-L167

Tool used

Manual Review , Foundry Testing

Recommendation

- handle the situation where `getTierPrizeSize(numberOfTiers - 3)` prize is `0`, to make sure that the prize for winner still can be claimed by claimers.

Discussion

elhajin

- This issue is distinct from 112 . In 112 , the issue is that having more winners in a specific tier than the prize count for that tier would lead to **insufficient liquidity** for the excess winners in case the reserve isn't enough to cover that . However, this is not true, as the protocol addresses this using the `tierUtilizationRatio` to manage such situations, indicating that this is a known limitation. Otherwise, there would be no point in having the `tierUtilizationRatio` variable .
- This issue is completely different. It addresses the scenario where the `lastTier` before the canary tiers has **0 liquidity**. In this situation, the claimer won't be able to claim any prize for all winners in this draw. As a result, winners in this draw won't receive their prizes.
- given the lost of prize for winners , and Dos of claiming process , this should be high severity

sherlock-admin3

> Escalate @nevillehuang This issue is distinct from #112 . In #112 , the issue is that having more winners in a specific tier than the prize count for that tier would lead to **insufficient liquidity** for the excess winners in case the reserve isn't enough to cover that . However, this is not true, as the protocol addresses this using the `tierUtilizationRatio` to manage such situations, indicating that this is a known limitation. Otherwise, there would be no point in having the `tierUtilizationRatio` variable .
>
> - This issue is completely different. It addresses the scenario where the `lastTier` before the canary tiers has **0 liquidity**. In this situation, the claimer won't be able to claim any prize for all winners in this draw. As a result, winners in this draw won't receive their prizes.

The escalation could not be created because you are not exceeding the escalation threshold.

You can view the required number of additional valid issues/judging contest payouts in your Profile page, in the Sherlock webapp.

0xjuaan

Escalate

On behalf of @elhajin

sherlock-admin3

> Escalate
>
> On behalf of @elhajin

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

WangSecurity

To clarify, the users indeed can receive rewards, the problem is that they have to do that by themselves and without claimers? But claimers are DOSed?

elhajin

Yes .. and the argument here is **this is not the expected behaviour**

- one of the main features of pooltogather is winners will receive prizes automatically through incentivized claimers and average users are not expected to check each draw if they win or not and claim their prizes ..
- so Dos on claiming will lead to winners not receiving their prizes except for those who claim for themselves which is not the protocol intent

0x73696d616f

This is a design choice. They cap the fee to the the prize pool of the last non canary tier. If this is 0, then fine, new draws have to be awarded so there is enough fee (if users still want to claim, they can call it). If prizes are not claimed, the number of tiers will decrease, which will increase the fee for the next draw, as the number of tiers will be less, so each tier has more liquidity and the number of prizes for the fee tier (number - 3) decreases, so the fee increases. So basically the transaction reverting or not does not matter because the max fee is 0, so if they want it to go through, they need to set the fee to 0.

Think about it this way, if there are no contributions, or small contributions in the last draw (which is the requirement for this), the fee should also be 0 or small.

elhajin

Good point .. but i disagree that this is a designe choice

- so in this case winners of tiers that are less then (numberOfTiers - 3) in this draw will lose their prizes due to **lack of incentive for claimers**.

- this is still an issue. *Claimers should always be incentivized to claim prizes for winners in each draw regardless of last tier liquidity* Since the protocol intent is winners get their prizes automatically.

Here a quote from the docs :

> Prizes expire every Draw. For daily prizes, that means they expire every 24 hours. **Incentivizing claimers is critical to ensure that users get their prizes**

- the decrease of tiers in the next draw is irrelevant to this issue.. cause winners of previous draw already lost their prizes

  If this is 0, then fine, new draws have to be awarded so there is enough fee

- this is not fine winning a prize for a user is the main idea of the protocol .. and if he lost his prize it's high likely he won't win it again in the new awarded draws

0x73696d616f

Claimers can not be incentivized if there was no liquidity provided, which was the design choice made by picking the last non canary tier prize size. The docs still match the code because the incentive is a % of something, if that something is 0, than the incentive is also 0.

What I meant by it's fine, is that users can claim prizes themselves, sure the protocol prefers that bots claim it, but this does not mean users can not claim them, even more so when there was no liquidity contributed. Bots will have incentives in the next draw.

Also, the fact that the transaction reverts when the prize is 0 has nothing to do with this, the issue itself is in how the protocol calculates the max fee, which is based on the liquidity contributed in the previous draws. When the max fee is 0, bots should claim with `feeRecipientZeroAddress` set to `0` to save gas, if they wish to do.

We can see that it is a design choice because the fix would be a different design of the fee, such as adding a fixed component, which is not a valid issue.

elhajin

I understand your POV.

- just to add to my previous comments other tiers (less then numberOfTiers -3) rely on contribution from several draws (up to contribution of a user in last 365 draw for grand prize) to determine the prize size and winner .and my POV is Not incentive claimer to claim those prizes because one of those draws has small contribution is an issue.

To summarize that for the judge:

- there is no incentives for claimers to claim prizes for winners when last tier liquidity equals 0
- this will lead to users who rely on the fact that prizes are distributed automatically lose their prizes

0x73696d616f

It is a design choice and another proof is that the issue will never happen due to

1) utilization ratio,
2) requirement of claiming all prizes in the past draw in the canary tiers (and in some cases the last non canary tier, tier 4 to tier 4 and tier 5 to tier 4),
3) not having contributions for a whole draw
4) the shares of the last non canary tier are much bigger than the canary ties, so whatever liquidity is left is mostly sent to it.

Starting state, draw 5 4 tiers, canary tiers 1 and 2 are claimed 50% due to utilization ratio

Final state, draw 6 *Example1*, tiers move from 4 to 5 even if 0 contributions in this draw, the previous canary tiers were only claimed 50%, so there is liquidity to distribute to tier 2 (the one that leads to the max fee), and most of it is sent there.

Example2, tiers stay at 4 even if 0 contributions, the last canary tiers will be distributed, and tier 1 (the max fee), even if the expected prizes were claimed for tier 1, still remains 50% liquidity + most of the amount from the canaries due to having more shares.

Starting state, draw 5 5 tiers, tier2, canary tiers 1 and 2 are claimed 50% due to utilization ratio

Final state, draw 6 tiers move down to 4 tier2 and the canaries get redistributed, they were only claimed 50% due to the utilization ratio, so tier 1 (the max fee) will always have liquidity, even if no contributions occur and it receives most of the liquidity.

So utilization ratio + requirement to claim all prizes + not having contributions in the whole draw + last non canary tier having much more shares than canary tiers make this issue impossible to happen

elhajin

Will never happen???

- I think you mixed things up; utilization ratio doesn't prevent the claiming of all the liquidity of a tier since we can have more winners than the prize count of a tier

- utilization ratio and last tiers before canary have more prizes can also be a helping factors to this issue to be presented due to high liquidity fragmentation over all prizes then scaling down by utilisation ratio that rounds the prize to zero
- not having contribution in a draw is high likely as vaults doesn't accumulate yeild daily to contribute it
- I'd love the sponsor to leave his comment here

0x73696d616f

> I think you mixed things up; utilization ratio doesn't prevent the claiming of all the liquidity of a tier since we can have more winners than the prize count of a tier

It decreases the likelihood a lot as it needs to outperform the expected claim counts by a big margin (double) on both canary tiers and the last non canary tier in some cases (tier 4 to tier 4 and tier 5 to tier 4). Assume 2 users, each 0.5 contribution to prize, probably of claiming both canaries twice is `0.5*0.5*0.5*0.5` = 0.0625 (simplifying 1 prize count per canary tier, doesnt matter having more in terms of expected probabilities, they always have to overperform by twice the amount).

> utilization ratio and last tiers before canary have more prizes can also be a helping factors to this issue to be presented due to high liquidity fragmentation over all prizes then scaling down by utilisation ratio that rounds the prize to zero

This doesnt matter, liquidity ratio just decreases 50% here, and the prize count does not get big enough to send to 0, even if we are in tier 11 (highly unlikely), 4^8 == 65536. The prize token is WETH, which has 18 decimals, 65536 / 1e18 / 0.5 == 1.31072e-13 WETH, which is a dust contribution, enough for this to not happen.

> not having contribution in a draw is high likely as vaults doesn't accumulate yeild daily to contribute it

It would require **all** vaults to not contribute in a day, which is highly unlikely.

So if we multiply all these chances, as they all need to happen simultaneously (and the fact that the last non canary tier has more shares, so it will absorve most liquidity), this will never happen. And it is a design choice regardless, but wanted to get this straight.

WangSecurity

Firstly, I can confirm that it's not a design decision.

Secondly, I agree that it's not a duplicate of #112.

Thirdly, based on the discussion above, this scenario indeed has significant requirements and constraints (based on this and this comments), moreover, it only

DOSes the claimers, but the prizes can still be received. I understand that the goal is to claimers to claim prizes, still the prizes are not locked completely and can be claimed.

Planning to accept the escalation and de-dup the report.

elhajin

- Odds for canary tiers are 100%, and the claimer is getting the whole prize of those tiers. This means it matters if the prize for canary tiers is more than the gas the claimer will spend to claim it. If so, the claimer will claim canary tiers, and it's highly likely that they will find a **lot** of winners for these tiers (even exceeding the liquidity those canary tiers have when claiming for them all) given the high odds.
- When can this happen?
  - This can happen when we have a large enough contribution in the previous draw since liquidity for canary tiers is redistributed each draw.
- What happens when claimers claim all liquidity in canary tiers?
  - The next draw will increase the number of tiers, and the new tier (newNumberOfTiers - 3) will get its liquidity from the contribution of this draw. So if there were no or small contributions in this draw, this tier will get 0 liquidity.
- Why is it likely to get no contribution in the next draw while we got large enough liquidity in the previous draw?
  - As mentioned in my comment above, vaults don't accumulate yield daily, and it's highly likely if they contributed yesterday, they will have no yield to contribute today.

  It would require all vaults to not contribute in a day, which is highly unlikely.

It's highly likely. We're talking about two vaults in production at the current time (even with 10 vaults, it's highly likely this will happen). Whether a design choice or not, let's let the judge decide.

0x73696d616f

@elhajin as the utilization ratio is 50%, the odds of claiming all canary tiers twice are close to `0.0625`. 2 vaults is a really small number, it is expected to be more. Each vault has its own token, and more than 1 vault can exist for each token. So this number is likely to grow a lot. And keep in mind that these 2 events must happen at the same time, so the chance becomes really really small.

If we think there is a 1% chance no contributions happen in a day (this number seems too big, I think the chance is even smaller, in normal operating conditions, as

bots are incentivized to liquidate daily, if they don't, this is irregular), the chance would be `0.0625*0.01 = 0.0625%`, very small.

And even if this happens (highly unlikely), users can still claim prizes for themselves, or even the protocol may choose to sponsor it.

elhajin

- Again, I still think it's highly likely to get no contribution in a day.
- For example, if canary tiers are t2 and t3, each address (user) has chance to win up to : **4^2+ 4^3 = 80** prizes and we have **80*2(UR) = 160** prizes. I'm curious how you calculated the odds for claiming canary tiers given that we don't know how many users contributed to vaults and how many vaults contributed to the PrizePool in this draw?

  As the utilization ratio is 50%, the odds of claiming all canary tiers twice are close to 0.0625.

0x73696d616f

> Again, I still think it's highly likely to get no contribution in a day.

It's not the expected, regular behaviour. The protocol incentivizes liquidating daily, so anything other than this is unlikely. In this case, there are multiple vaults, even more unlikely. It would require having unexpected behaviour (not liquidating daily) for all vaults, which is very unlikely.

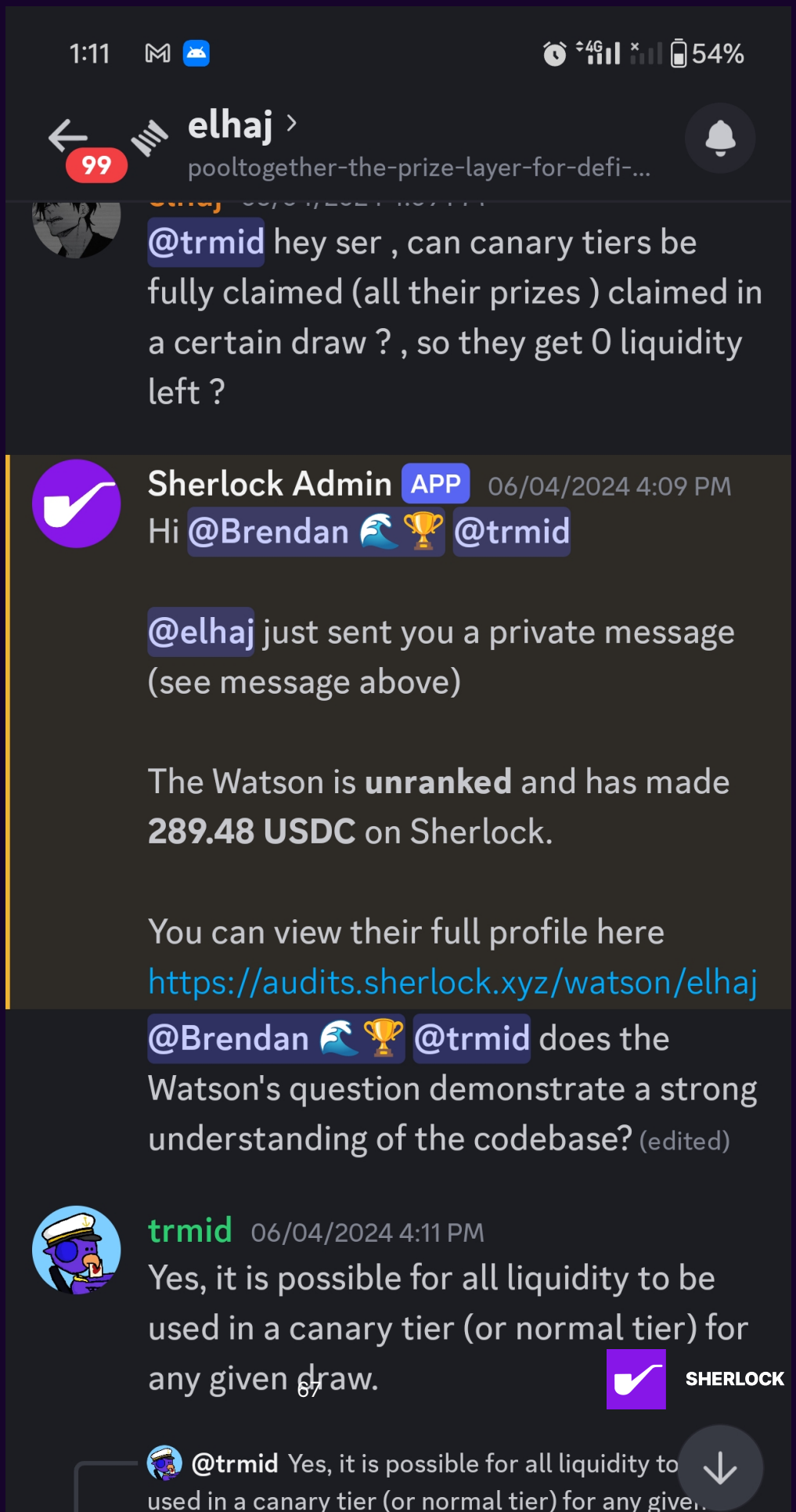
About the odds, here is the calculation. Assuming 1 prize for the canary tiers, we need at least 2 users to get 4 prizes (if there are 2 users, they at most get 2 prizes, so this issue does not exist). If the users have each 50% of the vault contribution, than their chance to win is 50%, as canary tiers have 100% odds. So as each user has a 50% chance of winning, and we would need them to win both prizes each, which is `0.5*0.5*0.5*0.5`. The calculations just get more complicated math wise with more prizes, key is they have to overperform the system, which is very unlikely. Using a claim count of 1 gives a good idea of the likelihood.

elhajin

> It's not the expected, regular behaviour. The protocol incentivizes liquidating daily, so anything other than this is unlikely. In this case, there are multiple vaults, even more unlikely. It would require having unexpected behaviour (not liquidating daily) for all vaults, which is very unlikely

- that's not true and you can check on-chain the deployed version of the protocol
- your calculation shows the probability of 6.25%(0.0625) that all canary tiers will be claimed .. which is for me high enough

- moreover I spoke about that with sponsor during the contest and he


1:11
54%
99
elhaj
pooltogether-the-prize-layer-for-defi-...
@trmid hey ser , can canary tiers be fully claimed (all their prizes) claimed in a certain draw ? , so they get 0 liquidity left ?
Sherlock Admin APP 06/04/2024 4:09 PM
Hi @Brendan 🌊🏆 @trmid
@elhaj just sent you a private message (see message above)
The Watson is unranked and has made 289.48 USDC on Sherlock.
You can view their full profile here https://audits.sherlock.xyz/watson/elhaj
@Brendan 🌊🏆 @trmid does the Watson's question demonstrate a strong understanding of the codebase? (edited)
trmid 06/04/2024 4:11 PM
Yes, it is possible for all liquidity to be used in a canary tier (or normal tier) for any given draw.
SHERLOCK
@trmid Yes, it is possible for all liquidity to
used in a canary tier (or normal tier) for any given

0x73696d616f

@elhajin, the sponsor said it's possible to claim all prizes, which I agree with, but very unlikely, given the utilization ratio. But the fact that the vaults have to not contribute in the next day, further reduces the likelihood. The 2 chances multiplied are very low for this to be high, estimated at 0.0625% or less.

elhajin

I think we made our points clear .. let's let the judge decide that ser

0x73696d616f

And the expected behaviour part is true. Ok they only have 2 deployed vaults right now. But still the expected behaviour is them contributing daily. So the chances go from very low to non existent, considering winning all prizes by double the amount (0.0625) and not contributing in a single day. The actual chance is closer to 0.0625% or lower due to the 2 conditions, not just 6.25% as you mentioned.

0x73696d616f

Also, they have a lot of vaults deployed, 19, 12 and 11, respectively. https://optimistic.etherscan.io/address/0x0C379e9b71ba7079084aDa0D1c1Aeb85d24dFD39 https://arbiscan.io/address/0x44be003e55e7ce8a2e0ecc3266f8a9a9de2c07bc https://basescan.org/address/0xe32f6344875494ca3643198d87524519dc396ddf

WangSecurity

To clarify, the chances of this happening is ~0.0625% and not 6.25%, so when @0x73696d616f wrote just 0.0625 it meant 0.0625%, correct?

0x73696d616f

Yes 0.0625%, after seeing that there are so many deployed vaults, it's even lower than this.

elhajin

@WangSecurity how so ? it's 6%.. `50*50*50*50/(100*100*100*100) = 625/100` which is 6.25% !! It's basic math

0x73696d616f

@elhajin you are not multiplying by the chance of not liquidating in a whole day, which is very low as explained before. There are at least 12 vaults, so all vaults would have to behave unexpectedly and not liquidate in a day for this to happen. I attributed a chance of 1% to this scenario, which is very reasonable, in reality it will be lower.

@WangSecurity to be clear, for this issue to happen all the canary prizes have to be claimed, which is a `6.25%` chance **AND** no contributions from **all** vaults can happen in a whole day, so we need to multiply both probabilities. The protocol expects

vaults to liquidate daily and has a tpda mechanism in place to ensure this, it would be extremely unlikely for all vaults to not liquidate in a single day. Thus, we can use an upper bound chance of 1% for this event, but we can see how it is extremely unlikely. So 0.0625*0.01 = 0.000625 = 0.0625%

elhajin

- First, we are not talking about that; we are talking about the probability that Canary tiers will be claimed, which is indeed 6.25%.
- Second, you're stating some of your thoughts as facts (at least 12 vaults, contributing daily is the expected behavior , 1% no contribution), which I believe are not true.
- Third, we don't know which vaults will be active (depending on users adopting them) and which will not. So, even with 100 vaults deployed, it's irrelevant if they are not active(anyone can deploy a vault).
- When I was talking about on-chain activity, I meant how many vaults are contributing, not how many are deployed. And as you can see here, **we have only 1 contribution in 69 days** (I understand that the V5 is still new).

0x73696d616f

> First, we are not talking about that; we are talking about the probability that Canary tiers will be claimed, which is indeed 6.25%.

This is false, we need to multiply by the chance of all vaults not contributing in a day, @WangSecurity if u need any further explanation let me know but this is factual.

> Second, you're stating some of your thoughts as facts (at least 12 vaults, contributing daily is the expected behavior , 1% no contribution), which I believe are not true.

The chance of all vaults behaving in a way they are not incentivized to is extremely unlikely. We can use 1%, but it is likely lower than this. If we say each vault has 10% chance of not contributing in a day, the chance would be 0.1^12=1e-12, with 12 vaults. With 2 vaults, it's 0.1^2 = 0.01 = 1%. So the chance goes from very low to impossible. Also, if we have less vaults, the likelihood of not contributing in a day will decrease for each because there is way more liquidity to liquidate, so the incentive is huge.

> Third, we don't know which vaults will be active (depending on users adopting them) and which will not. So, even with 100 vaults deployed, it's irrelevant if they are not active.

With regular user operation it can be expected a decent number of vaults will be active, anything other than this is extremely unlikely and it's not what the protocol expects.

> When I was talking about on-chain activity, I meant how many vaults are contributing, not how many are deployed. And as you can see here, we have only 1 contribution in 69 days (I understand that the protocol is still new).

We need to talk about regular conditions, not some bootstrap phase. If you want to consider weird activity, than we need to consider the prize may be very small, this phase has a low likelihood of happening, bots may choose not claim all prizes due to low liquidity, so on and so forth. In phases like this bots may not even be setup so this issue will not happen anyway.

This issue is an extremely edge case which will never happen, even medium is questionable. The stars would have to align for this to happen under normal operations.

Hash01011122

> The chance of all vaults behaving in a way they are not incentivized to is extremely unlikely. We can use 1%, but it is likely lower than this. If we say each vault has 10% chance of not contributing in a day, the chance would be 0.1^12=1e-12, with 12 vaults. With 2 vaults, it's 0.1^2 = 0.01 = 1%. So the chance goes from very low to impossible. Also, if we have less vaults, the likelihood of not contributing in a day will decrease for each because there is way more liquidity to liquidate, so the incentive is huge.

H/M severity don't take likelihood, only impact and constraints. REFERENCE If invalidation reason for this is likelihood then issue #88 should be invalid too

WangSecurity

Firstly, I believe the constraints for this scenario to happen are extremely high (all the canary prizes have to be claimed, which is a 6.25% chance AND no contributions from all vaults can happen in a whole day). Secondly, the prizes are not lost per se, it's just the claimers who are DOSed. I understand that's an important part of the protocol, but still the funds are not locked, and winners can claim the prizes themselves.

I believe based on these two points together, low severity is indeed more appropriate. The decision remains the same, planning to accept the escalation and de-dup this issue as invalid.

Hash01011122

Wrapping around my head on how this is not breakage of core functionality.

> the prizes are not lost per se, it's just the claimers who are DOSed. I understand that's an important part of the protocol, but still the funds are not locked, and winners can claim the prizes themselves.

WangSecurity

I agree this breaks the core contract functionality, but it has to have impact besides just breaking the core contract functionality:

> Breaks core contract functionality, rendering the contract useless or leading to loss of funds

As I've said there are no funds lost, since the users can still claim the prizes. But, on the other hand, the idea that just came to my mind is that, claimers have their own contract `Claimer` and in the case of this issue, the entire contract is useless, correct?

elhajin

@WangSecurity That's the idea . And claimer contract also in scope.

WangSecurity

With that, I still believe there is no loss of funds per se, because the prizes still can be claimed by the winners themselves. But, the core contract `Claimer` would useless in that case, since bots' attempts to claim prizes will revert. Hence, I believe it qualifies for the "Breaks core contract functionality, rendering the contract useless". If it's wrong then please correct me.

Planning to accept the escalation and make a new family of medium severity. Are there any duplicates?

Hash01011122

There are no dupes of this issue its unique finding.

WangSecurity

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xjuaan: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-claimer/pull/34

10xhash

Fixed in https://github.com/GenerationSoftware/pt-v5-claimer/pull/33 and https://github.com/GenerationSoftware/pt-v5-claimer/pull/34 Now the VRGDA is run with 100% maxFee as the auction's high price. In case the targetFee is 0, it is set to 1% of maxFee

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-9: DoSed liquidations as `PrizeVault::liquidatableBalanceOf()` does not take into account the `mintLimit` when the token out is the asset

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/88

Found by

0x73696d616f

Summary

`PrizeVault::liquidatableBalanceOf()` is called in `TpdaLiquidationPair::_availableBalance()` to get the maximum amount to liquidate, which will be incorrect when `_tokenOut` is the `asset` of the `PrizeVault`, due to not taking the minted yield fee into account. Thus, it will overestimate the amount to liquidate and revert.

Vulnerability Detail

`TpdaLiquidationPair::_availableBalance()` is called in `TpdaLiquidationPair::swapExactAmountOut()` to revert if the amount to liquidate exceeds the maximum and in `TpdaLiquidationPair::maxAmountOut()` to get the maximum liquidatable amount. Thus, users or smart contracts will call `TpdaLiquidationPair::maxAmountOut()` to get the maximum amount out and then `TpdaLiquidationPair::swapExactAmountOut()` with this amount to liquidate.

> Compute how much yield is available using the maxAmountOut function on the Liquidation Pair. This function returns the maximum number of tokens you can swap out.

However, this is going to revert whenever the minted yield fee exceeds the mint limit, as `PrizeVault::liquidatableBalanceOf()` does not consider it when the asset to liquidate is the asset of the `PrizeVault`. Consider `PrizeVault::liquidatableBalanceOf()`:

```
function liquidatableBalanceOf(address _tokenOut) external view returns
↪  (uint256) {
    ...
    } else if (_tokenOut == address(_asset)) { //@audit missing yield percentage
↪  for mintLimit
        // Liquidation of yield assets is capped at the max yield vault withdraw
↪  plus any latent balance.
```

```
        _maxAmountOut = _maxYieldVaultWithdraw() +
↪   _asset.balanceOf(address(this));
    }
    ...
}
```

As can be seen from the code snipped above, the minted yield fee is not taken into account and the mint limit is not calculated. On `PrizeVault::transferTokensOut()`, a mint fee given by `_yieldFee = (_amountOut * FEE_PRECISION) / (FEE_PRECISION - _yieldFeePercentage) - _amountOut;` is always minted and the limit is enforced at the end of the function `_enforceMintLimit(_totalDebtBefore, _yieldFee);`. Thus, without limiting the liquidatable assets to the amount that would trigger a yield fee that reaches the mint limit, liquidations will be DoSed.

Impact

DoSed liquidations when the asset out is the asset of the `PrizeVault`.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/main/pt-v5-vault/src/PrizeVault.sol#L693-L696

Tool used

Manual Review

Vscode

Recommendation

The correct formula can be obtained by inverting `_yieldFee = (_amountOut * FEE_PRECISION) / (FEE_PRECISION - _yieldFeePercentage) - _amountOut;`, leading to:

```
function liquidatableBalanceOf(address _tokenOut) external view returns
↪   (uint256) {
    ...
    } else if (_tokenOut == address(_asset)) {
        // Liquidation of yield assets is capped at the max yield vault withdraw
↪   plus any latent balance.
        _maxAmountOut = _maxYieldVaultWithdraw() +
↪   _asset.balanceOf(address(this));
        // Limit to the fee amount
        uint256  mintLimitDueToFee = (FEE_PRECISION - yieldFeePercentage) *
↪   _mintLimit(_totalDebt) / yieldFeePercentage;
```

```
        _maxAmountOut = _maxAmountOut >= mintLimitDueToFee ? mintLimitDueToFee :
↪   _maxAmountOut;
    }
    ...
}
```

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> the proposed remediation is in fact what is already done in the return statement

nevillehuang

If mint limit is reached then liquidations are capped per comments

0x73696d616f

Escalate

This issue is valid because the docs state exactly how liquidations happen, and they will fail for some states, when they should go through.

> Compute how much yield is available using the maxAmountOut function on the Liquidation Pair. This function returns the maximum number of tokens you can swap out.

So we can agree there is DoS, as it will revert when the number of shares is close to the mint limit. Liquidations use the tpda mechanism, so the price picked and the bot that gets awarded depend on being extremely timely in the liquidation. Thus, we conclude that liquidations are time sensitive.

For these 2 reasons, this issue is valid.

sherlock-admin3

> Escalate
>
> This issue is valid because the docs state exactly how liquidations happen, and they will fail for some states, when they should go through.
>
> > Compute how much yield is available using the maxAmountOut function on the Liquidation Pair. This function returns the maximum number of tokens you can swap out.

> So we can agree there is DoS, as it will revert when the number of shares is close to the mint limit. Liquidations use the tpda mechanism, so the price picked and the bot that gets awarded depend on being extremely timely in the liquidation. Thus, we conclude that liquidations are time sensitive.
>
> For these 2 reasons, this issue is valid.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

WangSecurity

> which will be incorrect when _tokenOut is the asset of the PrizeVault

To clarify this line, you mean the asset of the underlying `yieldVault`, correct?

As I see the yield fee is taken here in the return statement of the function?

0x73696d616f

Yes `_tokenOut` is the asset of the underlying vault. The problem is that when this asset is picked, a fee is still minted and the mint limit is enforced, see here. But `liquidatableBalanceOf()` does not take into account that a fee is minted even if the asset picked is `_tokenOut`. You can see here that the mint limit is not checked. So it overestimates the maximum liquidatable balance and makes liquidations fail. The maximum liquidatable balance should be capped to take into account the fee, with the exact recommendation given in the issue.

InfectedIsm

I may have missed something @0x73696d616f, but isn't it what is done in the final return statement?

```
function liquidatableBalanceOf(address _tokenOut) external view returns
↪  (uint256) {
    uint256 _totalDebt = totalDebt();
    uint256 _maxAmountOut;
    if (_tokenOut == address(this)) {
        // Liquidation of vault shares is capped to the mint limit.
        _maxAmountOut = _mintLimit(_totalDebt);
    } else if (_tokenOut == address(_asset)) {
        // Liquidation of yield assets is capped at the max yield vault withdraw
↪  plus any latent balance.
        _maxAmountOut = _maxYieldVaultWithdraw() +
↪  _asset.balanceOf(address(this)); <----
    } else {
```

```
        return 0;
    }

    // The liquid yield is limited by the max that can be minted or withdrawn,
↪   depending on
    // `_tokenOut`.
    uint256 _availableYield = _availableYieldBalance(totalPreciseAssets(),
↪   _totalDebt);
    uint256 _liquidYield = _availableYield >= _maxAmountOut ? _maxAmountOut :
↪   _availableYield; <----

    // The final balance is computed by taking the liquid yield and multiplying
↪   it by
    // (1 - yieldFeePercentage), rounding down, to ensure that enough yield is
↪   left for
    // the yield fee.
    return _liquidYield.mulDiv(FEE_PRECISION - yieldFeePercentage,
↪   FEE_PRECISION); <----
}
```

We see that the return value of `liquidatableBalanceOf` is: `_maxAmountOut * (FEE_PRECISION - yieldFeePercentage) / FEE_PRECISION` (1)

So the amount that is returned by `liquidatableBalanceOf()` is reduced by the fee that will be taken.

And in fact, if you plug (1) as `_amountOut` into : (2) `_yieldFee = (_amountOut * FEE_PRECISION) / (FEE_PRECISION - _yieldFeePercentage) - _amountOut`

you get `_yieldFee = fee * _maxAmountOut` (3)

If we rewrite (1) in a more readable way we have `liquidatableBalanceOf = _maxAmountOut * ( 1 - fee)` (4)

And if you add (3) the fee amount + (4) _amountOut that will be sent to user, we get: `maxAmountOut * fee + maxAmountOut * (1 - fee)  ==  maxAmountOut * (fee + 1 - fee)  ==  maxAmountOut`

So everything seems to fit correctly ?

0x73696d616f

Yes, that part is correct, but the limit is not actually enforced, if you notice regardless of the mint limit, it always returns a value bigger than 0. Example, maxAmountOut = 10000 (there is 10000 yield to be liquidated) there are 10 shares left to the mint limit yield fee is 1% so it would return `10000*99/100` = 9900 then, the fee to mint is `9900*100/(100 - 1) - 9900` = 100 and 100 > 10, so it reverts

the correct max amount is, `(100 - 1)*10/1 = 990`, as mentioned in the issue if we do `990*100/(100 - 1) - 990` = 10, which is the max shares that can be minted

InfectedIsm

Got it thanks, so when the total prize vault shares is close to `TWAB_SUPPLY_LIMIT` , then it is possible that the fee to mint based on the maximum liquidatable underlying asset amount is greater than the `mintLimit = TWAB_SUPPLY_LIMIT - _existingShares`, thus causing a revert during the `_enforceMintLimit` call The liquidation pair `smoothingFactor` coupled with the relatively low value of the fee means that the vault must be really close the the `TWAB_SUPPLY_LIMIT` for this to happen, but when vault is near 100% of its capacity it can happen more frequently, or even make it impossible to liquidate when `TWAB_SUPPLY_LIMIT` is reached This could be addressed by manually setting the yield fees to 0 if it happens, though manual intervention would still be required.

WangSecurity

And last clarification, excuse me if a silly one, but want to confirm if it's correct:

This issue doesn't require the mintLimit to be reached, but to be very close to it? I see that it can happen even if it's not close to the limit, but will be more often?

0x73696d616f

It gets more likely the closer we are to the limit, as the value required to trigger it decreases. In the example here, the 990 tokens would not be liquidated, as it tries to liquidate way more and reverts.

WangSecurity

Thank you for that clarification, with that I agree that it's valid issue. Planning to accept the escalation and validate with medium severity.

Hash01011122

@WangSecurity Shouldn't this be low severity, as the likelihood of this to occur is very low.

Firstly, for this to occur is `_tokenOut` should be the asset `PrizeVault`,

Secondly, fee to mint based on the maximum liquidatable underlying asset amount is greater than `_enforceMintLimit`, thus causing a revert during the `_enforceMintLimit` call

Thirdly, Watson assumes that protocol wouldn't react if `maxAmountOut` and `_enforceMintLimit` happens to create this scenario which can be ensured when protocol manually sets the yield fees to 0 as pointed out @InfectedIsm

Likelihood of this to occur is very low and with no impact.

Also, check out @MiloTruck comment here: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/39#issuecomment-2182180107

0x73696d616f

@Hash01011122

> Firstly, for this to occur is _tokenOut should be the asset PrizeVault

This does not decrease the likelihood, it's a perfectly valid choice.

> Secondly, fee to mint based on the maximum liquidatable underlying asset amount is greater than _enforceMintLimit, thus causing a revert during the _enforceMintLimit call

This is the only requirement, but still means a considerable amount of tokens could not be liquidated. Without this requirement, it would be a high.

> Thirdly, Watson assumes that protocol wouldn't react if maxAmountOut and _enforceMintLimit happens to create this scenario which can be ensured when protocol manually sets the yield fees to 0 as pointed out @InfectedIsm

This changes nothing because the function is time sensitive so it's crucial for bots to ensure this does not revert. If the bot reverts due to this, it would not get the price itself, taking a loss and the price will be different.

> Likelihood of this to occur is very low and with no impact.

It's not very low as the limit may be reached and it causes DoS and is time sensitive (and loss of funds for users of the vault, as they would get less tokens in return, as the price would keep going down), hence medium is appropriate.

> Also, check out @MiloTruck comment here: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/39#issuecomment-2182180107

It has no relevance that another watson says very vaguely that the issue is low.

Hash01011122

> This changes nothing because the function is time sensitive

Any idea how much time sensitive it really is @0x73696d616f @trmid @nevillehuang??

0x73696d616f

@Hash01011122 yes, look at the way the tpda liquidation works, there are 2 reasons:

1. the price keeps decreasing, so if it is DoSed vault users will receive less prize tokens in return of the liquidation.

2. the first bot that calls the function to liquidate should get the profit, but it doesn't and the profit goes to someone else.

Hash01011122

@0x73696d616f I didn't quite understand what you commented, would you mind elaborating this one please.

0x73696d616f

@Hash01011122 no problem, the liquidation is performed by the liquidation pair. This liquidation pair computes the price here, `uint192 price = uint192((targetAuctionPeriod * lastAuctionPrice) / elapsedTime);`. The longer `elapsedTime`, the smaller the price becomes. So DoSing liquidations will decrease the price. This price, is how many tokens are sent to the prize pool as a contribution from the respective vault that is being liquidated. So by decreasing the price, the vault that is liquidated will get a lower contribution to the prize pool, decreasing prizes for all users of that vault.

For point 2, bots are racing in this game to try to liquidate in great conditions. Ideally, each bot wants a price as low as possible, but the catch is that if they wait too much, other bot may liquidate instead. Thus, as the first bot that tries to liquidate reverts, it will not get the prize, when it should.

MiloTruck

Going to add context as to why I think this is valid, but low severity.

Liquidations occur through `swapExactAmountOut()`, which essentially does the following:

- `swapAmountIn` is the amount of prize tokens that has to be transferred in. This is determined by `_computePrice()`, a linearly decreasing auction that decreases the price from infinity to near-zero over time.
- The liquidator specifies `_amountOut`, which is the amount of yield tokens he wants to receive.
- The liquidator pays `swapAmountIn` of prize tokens and receives `_amountOut` of yield tokens.

It's important to realize that `liquidatableBalanceOf()` is only used as the upper limit for `_amountOut`:

TpdaLiquidationPair.sol#L141-L144

```
uint256 availableOut = _availableBalance();
if (_amountOut > availableOut) {
    revert InsufficientBalance(_amountOut, availableOut);
}
```

This means that when a liquidator calls `swapExactAmountOut()`, he can specify anything from `0` to the value returned by `liquidatableBalanceOf()` multiplied by the smoothing factor (ie. `TpdaLiquidationPair.maxAmountOut()`).

The issue here describes how `liquidatableBalanceOf()` doesn't account for the mint limit, so it will return a liquidatable balance higher than the actual amount of yield tokens that can be transferred out. In this case, if `swapExactAmountOut()` is called with `_amountOut = maxAmountOut()`, it will revert as the issue has stated.

However, just because `swapExactAmountOut()` can revert when you pass a certain `_amountOut` value doesn't mean it will DOS liquidations. `liquidatableBalanceOf()` only returns the **maximum** liquidatable balance, so `swapExactAmountOut()` can always be called with a smaller `_amountOut` value so that it doesn't revert.

Assuming the vault is close to the mint limit, the scenario is:

- `liquidatableBalanceOf()` returns an inflated liquidatable balance.
- The "actual liquidatable balance" (ie. a value for `_amountOut` at which the mint limit won't be hit and `swapExactAmountOut()` passes) is smaller than `liquidatableBalanceOf()`. Note that this is the value `liquidatableBalanceOf()` would return if the recommendation was applied.

So what would happen is liquidators would just call `swapExactAmountOut()` with the actual liquidatable balance, such that the liquidation process doesn't revert. And the linearly decreasing auction mechanism ensures that a price for this actual liquidatable balance will always be found.

To lay it out simply, something like this would occur (assume the smoothing factor is 100% for simplicity):

1. `_computePrice()` decreases from infinity to a fair price for the liquidatable balance returned by `maxAmountOut()`.
2. Liquidator calls `swapExactAmountOut()` with `_amountOut = maxAmountOut()`, but realizes it reverts.
3. Liquidators waits for `_computePrice()` to decrease further until a fair price is reached for the vault's actual liquidatable balance.
4. Liquidator calls `swapExactAmountOut()` with `_amountOut` as the actual liquidatable balance, which doesn't revert.

The only impact here is that `maxAmountOut()` returns an inflated value, so liquidation bots calling `swapExactAmountOut()` in (2) waste gas. However, nearly all bots simulate their transactions beforehand, so the bot would simply realize that the transaction reverts and not send it.

You could argue that `maxAmountOut()` returns a wrong value and bots won't be able to figure out what `_amountOut` should be in (4), but this is equivalent to saying a

view function not used anywhere else in the code returning the wrong value is medium severity.

0x73696d616f

> Liquidator calls swapExactAmountOut() with _amountOut = maxAmountOut(), but realizes it reverts.

This is enough to cause problems, as it is an extremely time sensitive function. The price keeps going down and liquidations remain DoSed as it is a key function and part of the liquidation flow, explicitly stated by the protocol (it makes sense because bots want as much amount as possible so they will always call it before liquidating), so this is the source of truth, regardless of workarounds (which will not work as bots don't have much time to fix this, auctions last 6 hours, so they will not fix in time and the price drops too much)

> Compute how much yield is available using the maxAmountOut function on the Liquidation Pair. This function returns the maximum number of tokens you can swap out.

WangSecurity

Firstly, I agree that liquidation is a time-sensitive function. Secondly, as said in the docs, `maxAmountOut` is function that has to be called to determine the max amount that can be liquidated. With these two factors together, I believe this issue is indeed medium.

The decision remains the same, accept the escalation and upgrade severity to medium.

Also, @Hash01011122, H/M severity don't take likelihood, only impact and constraints.

0x73696d616f

> The decision remains the same, accept the escalation and leave the issue as it is.

Think you mean upgrade to medium

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-vault/pull/114

WangSecurity

@nevillehuang @0x73696d616f are there any duplicates?

WangSecurity

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0x73696d616f: accepted

nevillehuang

@WangSecurity Not as far as I know.

10xhash

Fixed. Now the amount is restricted such that the corresponding fee minted will fall within the mint limit

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-10: Estimated prize draws in TieredLiquidityDistributor are off due to rounding down when calculating the sum, leading to incorrect prizes

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/95

The protocol has acknowledged this issue.

Found by

0x73696d616f

Summary

Estimated prize draws in `TieredLiquidityDistributor` are off due to rounding down when calculating the expected prize count on each tier, leading to an incorrect next number of tiers and distribution of rewards.

Vulnerability Detail

`ESTIMATED_PRIZES_PER_DRAW_FOR_5_TIERS` and the other tiers are precomputed initially in `TieredLiquidityDistributor::_sumTierPrizeCounts()`. In here, it goes through all tiers and calculates the expected prize count per draw for each tier. However, when doing this calculation in `TierCalculationLib.tierPrizeCountPerDraw()`, `uint32(uint256(unwrap(sd(int256(prizeCount(_tier))).mul(_odds))));`, it rounds down the prize count of each tier. This will lead to an incorrect prize count calculation, for example:

```
grandPrizePeriodDraws == 8 days
numTiers == 5
prizeCount(tier 0) == 4**0 * 1 / 8 == 1 * 1 / 8 == 0.125 = 0
prizeCount(tier 1) == 4**1 * (1 / 8)^sqrt((1 + 1 - 3) / (1 - 3)) == 4 *
↪  0.2298364718  0.92 = 0
prizeCount(tier 2) == 4**2 * 1 == 16
prizeCount(tier 3) == 4**3 * 1 == 64
total = 80
```

However, if we multiply the prize counts by a constant and then divide the sum in the end, the total count would be 81 instead, getting an error of 1 / 81 1.12 %

Impact

The estimated prize count will be off, which affects the calculation of the next number of tiers in PrizePool::computeNextNumberOfTiers(). This modifies the whole rewards distribution for the next draw.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/main/pt-v5-prize-pool/src/abstract/TieredLiquidityDistributor.sol#L574
https://github.com/sherlock-audit/2024-05-pooltogether/blob/main/pt-v5-prize-pool/src/libraries/TierCalculationLib.sol#L113-L115

Tool used

Manual Review

Vscode

Recommendation

Add some precision to the calculations by multiplying, for example, by `1e5` each count and then dividing the sum by `1e5`.
`uint32(uint256(unwrap(sd(int256(prizeCount(_tier)*1e5)).mul(_odds))));` and `return prizeCount / 1e5;`.

Discussion

sherlock-admin3

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> rounding isn't an issue by itself it looks like a design choice

nevillehuang

Request PoC to facilitate discussion

Sponsor comments:

> Quality assurance at best, rounding errors are perfectly acceptable here and have no significant impact

What is the maximum impact here?

sherlock-admin4

PoC requested from @0x73696d616f

Requests remaining: **3**

0x73696d616f

Hi @nevillehuang, some context first: Rewards are assigned based on the max tier for a draw, starting at tier 4. So users can claim rewards on tiers 0, 1, 2, 3. Tier 5 has tiers 0, 1, 2, 3, 4. And so on. Each prize tier is claimed 4^t numbers of times for each user, where t is the tier. Each tier has certain odds of occuring, according to the formula here. The prize is distributed by awarding each tier pro-rata `tierShares` and canary tiers (the last 2) receive `canaryShares`. So, the more tiers, the more the total prize pool is distributed between tiers, decreasing the prize of each tier.

Now some specific example: When we go from max tier 4 to tier 5, the prize pool is dilluted to the new tier, decreasing the rewards for users for each tier (there are more shares). Going to tier 5 also means that there is 1 tier that has less odds of being claimed. Tier 1 is no longer awarded every draw. Thus, as we have dilluted the prize in the tiers, and added a tier with less odds of occuring, the total rewards expected are now less. This mechanism works to offset luck in the draw. If a certain draw is claimed too many times compared to the expected value, it will likely increase the max tier in the next draw, and decrease the expected rewards of the next draw.

And finally, the issue: The expected number of prizes collected on a certain max tier, ex 4, is calculated by summing the expected prizes collected on each tier (this is well explained in the issue). As the expected number will be off, the expected claim count will also be off. In the issue example, tier 5 has an incorrect expected value of 80 instead of 81. If the number of claimed prizes is 81, > 80, it will advance to tier 5. However, if the expected value was correctly calculated as 81, we would not advance to tier 5, but remain in tier 4. Consequently, the next draw will award less rewards, on average, than it should, as it incorrectly increased the max tier. Here is the calculation of the expected rewards being decreased for the next round, based on the numbers above:

```
prizePool == 1000
numShares = 100
canaryTierShares = 5

// tier 4
totalShares = 210
tierValue = 1000*100/210 = 476
canaryValue = 1000*5/210 = 24

476*0.125 + 476*1 + 24 = 560

// tier 5
totalShares = 310
tierValue = 1000*100/310 = 323
```

```
canaryValue = 1000*5/310 = 16

323*0.125 + 323*0.23 + 323*1 + 16 = 454
```

trmid

The existing algorithm returns the expected number of tiers based on claims. The calculation *informs* the rest of the system on the number of tiers. As long as the tiers can go up and down based on the network conditions, there is no unexpected consequence of the calculation rounding down.

MiloTruck

Escalate

I believe this issue should be low severity as there is barely any impact.

> This mechanism works to offset luck in the draw. If a certain draw is claimed too many times compared to the expected value, it will likely increase the max tier in the next draw, and decrease the expected rewards of the next draw.

This is only partially true. The primary purpose of increasing/decreasing the number of tiers based on the number of prizes claimed is to adapt to fluctuating gas prices due to network congestion. When gas prices are cheap, more prizes will be claimed than the expected amount. The pool adapts to this by moving up one tier, thereby decreasing the rewards per prize as it will still be profitable to claim prizes with low gas costs. The converse is also true for when gas prices are expensive.

`ESTIMATED_PRIZES_PER_DRAW_FOR_5_TIERS` and the other values are an approximation of the expected number of prizes claimed; they don't have to be the *exact* values calculated. As long as they are roughly around their exact values, which they are in this case, the number of tiers will still increase/decrease to adapt to changing gas prices.

I believe this is what the sponsor means by "As long as the tiers can go up and down based on the network conditions, there is no unexpected consequence".

> The estimated prize count will be off, which affects the calculation of the next number of tiers in `PrizePool::computeNextNumberOfTiers()`. This modifies the whole rewards distribution for the next draw.

This is an over-exaggeration of the impact. There is no "correct" rewards distribution for the next draw based on the number of prizes claimed - as long as the rewards distribution scales to accommodate fluctuating gas prices, the mechanism has fulfilled its purpose and there is no issue.

sherlock-admin3

> Escalate

I believe this issue should be low severity as there is barely any impact.

> This mechanism works to offset luck in the draw. If a certain draw is claimed too many times compared to the expected value, it will likely increase the max tier in the next draw, and decrease the expected rewards of the next draw.

This is only partially true. The primary purpose of increasing/decreasing the number of tiers based on the number of prizes claimed is to adapt to fluctuating gas prices due to network congestion. When gas prices are cheap, more prizes will be claimed than the expected amount. The pool adapts to this by moving up one tier, thereby decreasing the rewards per prize as it will still be profitable to claim prizes with low gas costs. The converse is also true for when gas prices are expensive.

`ESTIMATED_PRIZES_PER_DRAW_FOR_5_TIERS` and the other values are an approximation of the expected number of prizes claimed; they don't have to be the *exact* values calculated. As long as they are roughly around their exact values, which they are in this case, the number of tiers will still increase/decrease to adapt to changing gas prices.

I believe this is what the sponsor means by "As long as the tiers can go up and down based on the network conditions, there is no unexpected consequence".

> The estimated prize count will be off, which affects the calculation of the next number of tiers in `PrizePool::computeNextNumberOfTiers()`. This modifies the whole rewards distribution for the next draw.

This is an over-exaggeration of the impact. There is no "correct" rewards distribution for the next draw based on the number of prizes claimed - as long as the rewards distribution scales to accommodate fluctuating gas prices, the mechanism has fulfilled its purpose and there is no issue.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0x73696d616f

@MiloTruck there is a correct expected number of prizes, based on the odds and prize count for each tier, and this issue proves it will be off due to rounding down, when it could be fixed (at least partially, not rounding down more than 1 unit). The comments are the source of truth, and the expected number is off

> /// @notice Computes the expected number of prizes for a given number of tiers.

The mentioned impact is true.

So unless a comment in the code or readme information is found that says rounding down more than 1 unit is okay, this issue is valid.

WangSecurity

> As long as they are roughly around their exact values, which they are in this case, the number of tiers will still increase/decrease to adapt to changing gas prices

So in the example by @0x73696d616f if the expected number is 81, but the actual one is 80, the number of tiers can still increase/decrease based on gas conditions?

> If the number of claimed prizes is 81, > 80, it will advance to tier 5. However, if the expected value was correctly calculated as 81, we would not advance to tier 5, but remain in tier 4.

Not sure if I understand it correctly, if expected is 80 and actual is 81, we increase to tier 5. If expected is 81 and actual is 81, we remain tier 4?

0x73696d616f

To advance we need `actual >= expected`, because if `actual < expected` it returns 4. And it also impacts the gas adaptation mechanism because the expected value is off.

MiloTruck

> So in the example by @0x73696d616f if the expected number is 81, but the actual one is 80, the number of tiers can still increase/decrease based on gas conditions?

@WangSecurity My point is that the number of tiers will only need one more claim to increase, and there isn't really a difference between 80 claims VS 81 claims.

There's a reason why the sponsor chose to dispute, and I believe it's because as long as the number of tiers increases at some point, regardless of at 80 or 81, the number of tiers will still increase/decrease based on gas conditions.

0x73696d616f

it was not said anywhere that it is okay for the expected number to be off, so the hierarchy of truth holds and the 2 mentioned impacts are correct. The point is that the difference exceeds small amounts, as is the case in this issue, where the shutdown still "works", but not in the right draw.

MiloTruck

> it was not said anywhere that it is okay for the expected number to be off, so the hierarchy of truth holds and the 2 mentioned impacts are correct. The point is that the difference exceeds small amounts, as is the case in this issue, where the shutdown still "works", but not in the right draw.

I don't see how #28 is relevant here, it's a completely different issue.

Essentially it boils down to whether you believe the difference between 80 vs 81 claims is significant enough such that it affects the mechanism's ability to increase/decrease tiers according to gas prices. You believe the difference is significant while I (and I believe the sponsor?) don't. Will leave it to the judges to decide though, ultimately it's not up to us.

InfectedIsm

https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/95#issuecomment-2164066705

> If the number of claimed prizes is 81, > 80, it will advance to tier 5. However, if the expected value was correctly calculated as 81, we would not advance to tier 5, but remain in tier 4.

So, this simply shift the tier advancement to number of prize > 80 rather than 81 due to rounding in computation (which again isn't an issue, it can even sometimes used purposefully) The formula that the devs have chosen is arbitrary, as much as how they decided to implement it. They could have gone for any other formula to decide the limit of the tier expansion, which would have given another limit, could have been `t^5 + 1` or `t ^2 * 2 + t^4`, the main idea is to expand/contract the number of tier based by measuring how far from the expected odds of winning we are.

0x73696d616f

The formula they picked is not arbitrary, it's the exact expected number of claims given that each prize has certain odds and claim count. It will be off due to rounding down.

WangSecurity

> The primary purpose of increasing/decreasing the number of tiers based on the number of prizes claimed is to adapt to fluctuating gas prices due to network congestion. When gas prices are cheap, more prizes will be claimed than the expected amount. The pool adapts to this by moving up one tier, thereby decreasing the rewards per prize as it will still be profitable to claim prizes with low gas costs

But in that case, if the expected number is off (not the one that should be due to rounding down), then the system won't adapt to fluctuating gas prices correctly and won't move up one tier when it needs to. Or it's wrong?

I see that the rounding down is only by 1, but still in the edge case, expressed in the report, it lead to the protocol behaving in the incorrect way.

Please tell me where I'm wrong.

And is it said anywhere that this expected number doesn't have to be the exact one?

And a bit more info, sponsors setting "sponsor confirmed/disputed" and "won't/will fix" labels doesn't affect the final decision.

0x73696d616f

@WangSecurity it will take 1 less claim than supposed to advance to the next tier, so the gas mechanism will not work as expected, you're right.

I could not find any mention that the expected number can be off and as the error exceeds small amounts, it's valid.

WangSecurity

With that, I agree that it's a valid issue cause if the expected number is incorrect, then the protocol cannot correctly adapt to gas prices, hence, won't expand tiers correctly, when it should.

Planning to reject the escalation and leave the issue as it is.

Hash01011122

I agree with @MiloTruck and @InfectedIsm reasoning, this issue should be informational.

There's no impact of this issue as pointed out by sponsors:

> The existing algorithm returns the expected number of tiers based on claims. The calculation informs the rest of the system on the number of tiers. As long as the tiers can go up and down based on the network conditions, there is no unexpected consequence of the calculation rounding down.

0x73696d616f

@Hash01011122 there is impact in the rewards distribution, and it was not documented that it's fine for the expected number to be incorrect.

Hash01011122

@0x73696d616f with all due respect, I don't see this as a valid issue.

0x73696d616f

The reason you provided is based on information that was added post contest (that this is an acceptable bug), which has no weight.

WangSecurity

I still stand by my point stated above that in fact has impact and leads to the protocol adapting to gas prices incorrectly. The decision remains the same, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- MiloTruck: rejected

Issue M-11: `Claimers` can receive less `feePerClaim` than they should if some prizes are already claimed or if reverts because of a reverting hook

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/124

The protocol has acknowledged this issue.

Found by

0x73696d616f, hash, infect3d, jovi, zraxx

Summary

If a claimer propose an array of prizes to claim, but some of these prizes have already been claimed, or some claims revert, then the actual `feePerClaim` received will be less **compared to what it should really be, as expected by the VRGDA algorithm**

This happens because `_computeFeePerClaim` can undervaluate the value of `feePerClaim`, as it compute failed claims as successful ones.

This poses an issue, as `feePerClaim` is then used as an input by `_vault.claimPrize(_winners[w], _tier, _prizeIndices[w][p], _feePerClaim, _feeRecipient)`, which will transfer the undervaluated fee to the claimer.

Vulnerability Detail

Auctions for claiming prizes are based on the VRGDA algorithm In simple terms, this algorithm update price depending on either the numbers of claim is behind or ahead of time schedule. In order to have a schedule, a target of claim per time unit is defined. Just to give an idea, let's simplify that to the extreme (we will see complete formula afterward) and say : `price(t) = claim/expected * targetPrice(t)` E.g: if 10 claims per 2 hours are exepected, then at t=1h, 5 claims should be concluded. If only 4 were claimed, then we can calculate that (4 claims)/(5 expected) < 1, price will be lower that target. if 6 were claimed, then we will have (6 claims)/(5 expected) > 1, price will be greater than target.

The formula that has been implemented into `LinearVRGDALib` is the following:

```
price = p0 * e^ (k * (t - n+1/r))    ; with k = ln(maxFee/minFee) * t_target
```

With:

- `n` the number of claim already completed
- `r` the expected rate per hour
- `k` the decay constant (speed at which price will change)
- `p0` the target price (or fee in our case)

The more `k *(t - n+1/r) > 0`, the more `price > p0` When `t = n+1/r <=> (k * (t - n+1/r)) = 0`, then `price = p0` The more `k * (t - n+1/r) < 0`, the more `price < p0`

We understand that the more whe are behind schedule in term of expected claim, the higher the fees earned by claimer will be. And the more we are ahead of schedule, the lower the fee for claimers will be (as their is no urgency)

Scenario

Now let's see what happens in this scenario:

1. For now, 0 prizes have already been claimed from the prize pool, `n = 0`
2. Alice has 5 prizes to claim, she build her claim array
3. It seems that 2 of the prizes Alice was going to claim are claimed right before, now `n = 2`
4. Alice tx is executed with the 5 prizes

Now let's see what happen from a code perspective. The code above is the entry point for claiming prizes. As we can see L113, the `feePerClaim` is computed based on the number of claims to count (`_countClaims`) and the number of already claimed prizes (`prizePool::claimCount()`) Then, the computed `feePerClaim` value is given to `_claim` which actually claim the prizes into the Prize Pool.

https://github.com/sherlock-audit/2024-05-pooltogether//blob/main/pt-v5-claimer/src/Claimer.sol#L113

```
File: pt-v5-claimer/src/Claimer.sol

090:   function claimPrizes(
091:     IClaimable _vault,
092:     uint8 _tier,
093:     address[] calldata _winners,
094:     uint32[][] calldata _prizeIndices,
095:     address _feeRecipient,
096:     uint256 _minFeePerClaim
097:   ) external returns (uint256 totalFees) {
...:
...:         /* some code */
...:
```

```
112:     if (!feeRecipientZeroAddress) {
113:       feePerClaim = SafeCast.toUint96(_computeFeePerClaim(_tier,
↪  _countClaims(_winners, _prizeIndices), prizePool.claimCount()));
114:       if (feePerClaim < _minFeePerClaim) {
115:         revert VrgdaClaimFeeBelowMin(_minFeePerClaim, feePerClaim);
116:       }
117:     }
118:
119:     return feePerClaim * _claim(_vault, _tier, _winners, _prizeIndices,
↪  _feeRecipient, feePerClaim);
120:   }
```

Now, let's see how `_computeFeePerClaim` actually compute `feePerClaim`. We see above L230-241 that a fee is calculated for each of the claims of the array, starting at `_claimedCount` (The number of prizes already claimed) based on the VRGDA formula L309. The returned value (which is stored into `feePerClaim`) is the averaged fee as shown L241. And as we explained earlier, the higher the number of claim we make, the lower the earned fee are. So, a higher value of `_claimedCount + i` will give lower fees.

https://github.com/sherlock-audit/2024-05-pooltogether//blob/main/pt-v5-claimer/src/Claimer.sol#L236

```
File: pt-v5-claimer/src/Claimer.sol
205:   /// @param _claimCount The number of claims to check
206:   /// @param _claimedCount The number of prizes already claimed
207:   /// @return The total fees for the claims
208:   function _computeFeePerClaim(
209:     uint8 _tier,
210:     uint256 _claimCount,
211:     uint256 _claimedCount
212:   ) internal view returns (uint256) {
...:
...:       /* some code */
...:
227:     uint256 elapsed = block.timestamp -
↪  (prizePool.lastAwardedDrawAwardedAt());
228:     uint256 fee;
229:
230:     for (uint256 i = 0; i < _claimCount; i++) {
231:       fee += _computeFeeForNextClaim(
232:         targetFee,
233:         decayConstant,
234:         perTimeUnit,
235:         elapsed,
236:         _claimedCount + i,
```

```
237:           _maxFee
238:         );
239:       }
240:
241:       return fee / _claimCount;
242:     }
...:
...:         /* some code */
...:
301:     function _computeFeeForNextClaim(
302:       uint256 _targetFee,
303:       SD59x18 _decayConstant,
304:       SD59x18 _perTimeUnit,
305:       uint256 _elapsed,
306:       uint256 _sold,
307:       uint256 _maxFee
308:     ) internal pure returns (uint256) {
309:       uint256 fee = LinearVRGDALib.getVRGDAPrice(
310:         _targetFee,
311:         _elapsed,
312:         _sold,
313:         _perTimeUnit,
314:         _decayConstant
315:       );
316:       return fee > _maxFee ? _maxFee : fee;
317:     }
318:
```

What we can see from this, is that the computation will be executed for `_claimCount = 2` and up to `i = 5`, so as if there has been 7 claimed prizes, while in reality only 5 prizes are claimed, leading in an undervaluation of the fees to award. As you probably have infered, the computation should have been made for `i = 3` to be correct.

Impact

The `feePerClaim` computation is incorrect as the VRGDA is calculated for more claims that will really happen, leading to less fee earned by claimers at the time of the call.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether//blob/main/pt-v5-claimer/src/Claimer.sol#L113 https://github.com/sherlock-audit/2024-05-pooltogether//blob/main/pt-v5-claimer/src/Claimer.sol#L236

Tool used

Manual Review

Recommendation

The `PrizePool` contract expose a function to check if a prize has already been claimed: `wasClaimed` This can be used to countClaims based on the actual true number of claimable prizes from the array.

This isn't a "perfect" solution though, as there are still issues when not already claimed prizes revert because of reverting prize hooks. In that case, VRGDA will still count the claim as happening, but we can consider this less likely to happen.

```
  function _countClaims(
    address[] calldata _winners,
    uint32[][] calldata _prizeIndices
  ) internal pure returns (uint256) {
    uint256 claimCount;
    uint256 length = _winners.length;
    for (uint256 i = 0; i < length; i++) {
-     claimCount += _prizeIndices[i].length;
+     numPrize = _prizeIndices[i].length;
+     for(uint256 j = 0; j < numPrize; j++) {
+       bool wasClaimed = wasClaimed(_vault, _winner, _drawId,_tier,
 ↪  _prizeIndex);
+       if(!wasClaimed) {
+        claimCount += 1;
+       }
+     }
    }
    return claimCount;
  }
```

Discussion

trmid

The prize pool must be passed the fee during each individual claim call so it can give the claimer their reward. The fees are calculated before in bulk and then split evenly for each claim in the batch. As the issue demonstrates, it is possible for some of these claims to revert, resulting in less fees collected than if the claimer excluded the reverting claims.

It's important for the claimers to simulate the results before claiming to ensure their claims will result in the expected rewards. Claimers compete to perform claims at

the lowest costs, so the bots that simulate results will have higher success rates and be able to claim at lower margins compared to those who don't.

The impact of this issue is very low for competitive bots since claim simulations are commonly available and encouraged.

InfectedIsm

Hey @nevillehuang, just noticed #57 is incorrectly duplicated with this finding. The submission do not show the fee calculation error, it simply tell that a user could claim the canary tier before prizes are claimed, which will increase the PrizePool.claimedCount counter, which is expected behavior. There is no reference about already claimed prizes which are still counted as claimed by the VRGDA. Sorry for not having noticed it before.

InfectedIsm

@WangSecurity can you check this please, it has been overlooked, thanks (I've shared 6 days ago that #57 is not a duplicate of this finding)

Issue M-12: The RNG finish draw auction rewards are over-paid due to missing to account for the time it takes to fulfill the Witnet randomness request

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/126

The protocol has acknowledged this issue.

Found by

berndartmueller

Summary

The rewards for finishing the draw and submitting the previously requested randomness result are slightly overpaid due to the incorrect calculation of the elapsed auction time, wrongly including the time it takes to fulfill the Witnet randomness request.

Vulnerability Detail

The `DrawManager.finishDraw` function calculates the rewards for finishing the draw, i.e., providing the previously requested randomness result, via the `_computeFinishDrawReward` function in lines 338-342. The calculation is based on the difference between the timestamp when the start draw auction was closed (`startDrawAuction.closedAt`) and the current block timestamp. Specifically, a parabolic fractional dutch auction (PFDA) is used to incentivize shorter auction durations, resulting in faster randomness submissions.

However, the requested randomness result at block X is not immediately available to be used in the `finishDraw` function. According to the Witnet documentation, the randomness result is available after 5-10 minutes. This time delay is currently included when determining the elapsed auction time because `startDrawAuction.closedAt` marks the time when the randomness request was made, not when the result was available to be submitted.

Consequently, the protocol **always** overpays rewards for the finish draw. Over the course of many draws, this can lead to a significant overpayment of rewards.

It should be noted that the PoolTogether documentation about the draw auction timing clearly outlines the timeline for the randomness auction and states that finish draw auction price will rise **once the random number is available**:

The RNG auction for any given draw starts at the beginning of the following draw period and must be completed within the auction duration. Following the start RNG auction, there is an "unknown" waiting period while the RNG service fulfills the request.

***Once the random number is available**, the finished RNG auction will rise in price until it is called to award the draw with the available random number. Once the draw is awarded for a prize pool, it will distribute the fractional reserve portions based on the auction results that contributed to the closing of that draw.*

Impact

The protocol overpays rewards for the finish draw.

Code Snippet

pt-v5-draw-manager/src/DrawManager.sol#L339

```
311: /// @notice Called to award the prize pool and pay out rewards.
312: /// @param _rewardRecipient The recipient of the finish draw reward.
313: /// @return The awarded draw ID
314: function finishDraw(address _rewardRecipient) external returns (uint24) {
315:   if (_rewardRecipient == address(0)) {
316:     revert RewardRecipientIsZero();
317:   }
318:
319:   StartDrawAuction memory startDrawAuction = getLastStartDrawAuction();
...    // [...]
338:   (uint256 _finishDrawReward, UD2x18 finishFraction) =
↪  _computeFinishDrawReward(
339:   startDrawAuction.closedAt,
340:     block.timestamp,
341:     availableRewards
342:   );
343:   uint256 randomNumber = rng.randomNumber(startDrawAuction.rngRequestId);
```

Tool used

Manual Review

Recommendation

Consider determining the auction start for the finish draw reward calculation based on the timestamp when the randomness result was made available. This timestamp

is accessible by using the WitnetRandomnessV2.fetchRandomnessAfterProof function (instead of fetchRandomnessAfter). The `_witnetResultTimestamp` return value can be used to better approximate the auction start, hence, paying out rewards more accurately.

Discussion

sherlock-admin4

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> this is exactly the `_firstFinishDrawTargetFraction` goal

trmid

The suggested mitigation is flawed since it would return the timestamp that the RNG was generated on the Witnet network, not the timestamp that it was relayed to the requesting network.

The auction should be set with an appropriate ~~`_firstFinishDrawTargetFraction`~~ `_auctionTargetTime` value such that the estimated auction value is reached after the 5-10 min expected relay time. In addition, the `maxRewards` parameter on the `DrawManager` contract helps to ensure a reasonable max limit to the total auction payout in the case of an unexpected outage.

nevillehuang

@berndartmueller Could you take a look at the above comment? Does an appropriately admin set `_firstFinishDrawTargetFraction` mitigate this issue?

berndartmueller

Hey @nevillehuang!

> *The suggested mitigation is flawed since it would return the timestamp that the RNG was generated on the Witnet network, not the timestamp that it was relayed to the requesting network.*

This statement is mostly true, **but**, the timestamp can also be the `block.timestamp` at the time when the RNG request was relayed to the requesting network. Specifically, when reporters use the `reportResult` function, the timestamp is set to block.timestamp.

If reporters use one of the two other reporting methods, the timestamp is set to the time when the RNG was generated on the Witnet network (although it's not verified that reporters set this value exactly, the timestamp could also be set to block.timestamp). To be fair, on-chain Optimism transactions to the Witnet contract show that the `reportResultBatch` function is predominantly used.

However, reporters are free to use whatever relaying function they use, so it's possible that the timestamp is the time when the request was made available to the requesting network. Either way, IMO, this makes it a better approximation of when the random number is available to PoolTogether's `finishDraw`, than using the time when startDraw was called and the RNG requested (which neglects the time it needs for Witnet actors to relay the request to the Witnet chain, generate the random number and relay it back to the requesting chain)

> _The auction should be set with an appropriate *firstFinishDrawTargetFraction value such that the estimated auction value is reached after the 5-10 min expected relay time. In addition, the maxRewards parameter on the DrawManager contract helps to ensure a reasonable max limit to the total auction payout in the case of an unexpected outage*.

`_firstFinishDrawTargetFraction` is provided once in the `constructor` and stored in `lastFinishDrawFraction`. And `lastFinishDrawFraction` is updated in every finishDraw call. While it's true that this value is used to better estimate the expected rewards and the relay time, it's also not perfect. A single occurrence of a longer than usual RNG relay time is enough to set `lastFinishDrawFraction` to an outlier value (i.e., inflated value, but upper-capped by `maxRewards`).

Long story short, using the recommendation in this submission, in conjunction with providing a reasonable `_firstFinishDrawTargetFraction` value by the admin, further improves the accuracy of the rewards calculation (i.e., prevents overpaying).

trmid

Removed the "disputed" tag since this seems like a valid issue for the auctions, but may not need additional mitigation.

I realized I quoted the wrong parameter in my statement above:

> The auction should be set with an appropriate _firstFinishDrawTargetFraction value such that the estimated auction value is reached after the 5-10 min expected relay time.

I meant to say `_auctionTargetTime` here instead of `_firstFinishDrawTargetFraction`.

Issue M-13: Witnet is not available on some networks listed

Source:
https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/127

The protocol has acknowledged this issue.

Found by

0x73696d616f, 0xSpearmint1, AuditorPraise, trachev, volodya

Summary

Witnet is not deployed on Blast, Linea, ZkSync, Zerion, so draws will not be possible there. Vaults can still be deployed and yield earned, so it will fail when trying to start or complete draws.

Vulnerability Detail

The sponsor intends to know concerns when deploying the protocol to the list of blockchains in the readme.

> We're interested to know if there will be any issues deploying the code as-is to any of these chains

One of these issues is that Witnet is not supported on Blast, Linea, ZkSync or Zerion, so an alternative rng source must be used.

Prize vaults may still be deployed, accruing yield and sending it to the prize pool, but the prize pool will not be able to award draws and will only refund users when it shutdowns.

Impact

DoSed yield until the prize pool shutdown mechanism starts after the timeout.

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/main/pt-v5-prize-pool/src/PrizePool.sol#L938
https://docs.witnet.io/smart-contracts/supported-chains

Tool used

Manual Review

Vscode

Recommendation

Use an alternative RNG source on the non supported chains.

Discussion

sherlock-admin3

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> from project docs: ""The startDraw auction process may be slightly different on each chain and depends on the RNG source that is being used for the prize pool.""

trmid

We are in active communication with the Witnet team to ensure that a deployment is available before deploying a prize pool on the network. In the unlikely case that Witnet will not be available on the target network, an alternate RNG source can be integrated.

nevillehuang

@trmid If the contracts are deployed as it is, then the inscope rng-witnet contract cannot be deployed, so per the following contest details:

> We're interested to know if there will be any issues deploying the code as-is to any of these chains, and whether their opcodes fully support our application.

and the following sherlock guidelines:

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.

I believe this is a valid medium, if not it should have been made known as a known consideration

10xhash

Escalate

Witnet v2 is new. Even their documentation during the time of the contest was meant for V1 contracts clearly indicating that the integration is done considering

the future. One of the chains mentioned ie. Zerion is/was not even public during the time of competition

sherlock-admin3

> Escalate
>
> Witnet v2 is new. Even their documentation during the time of the contest was meant for V1 contracts clearly indicating that the integration is done considering the future. One of the chains mentioned ie. Zerion is/was not even public during the time of competition

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

InfectedIsm

> Escalate
>
> Witnet v2 is new. Even their documentation during the time of the contest was meant for V1 contracts clearly indicating that the integration is done considering the future. One of the chains mentioned ie. Zerion is/was not even public during the time of competition

Also I reiterate that the documentation is crystal clear regarding the fact that this is known by the devs :

> The startDraw auction process **may be slightly different on each chain and depends on the RNG source that is being used for the prize pool**. For example, Witnet is used on the Optimism deployment and can be started by calling the startDraw function on the RngWitnet contract associated with that deployment. Check the deployment contracts for each chain to see which RNG source is being used.

clearly indicating that the devs are well aware that the source will depend on the chain they deploy. This is like reading the doc saying "we know that we will have to select the RNG source depending on the chain we will deploy" and telling the devs "hey devs, be careful there are chain where Witnet is not available", I think we can do better than that.

nevillehuang

By hierachy of truth escalations should be rejected.

> If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules.

WangSecurity

I agree with the lead judge here and this warrants medium severity based on this comment. I see that the protocol's docs say "The startDraw auction process may be slightly different on each chain and depends on the RNG source that is being used for the prize pool", but based on README this issue has to be valid.

Planning to reject the escalation and leave the issue as it is.

10xhash

> I agree with the lead judge here and this warrants medium severity based on this comment. I see that the protocol's docs say "The startDraw auction process may be slightly different on each chain and depends on the RNG source that is being used for the prize pool", but based on README this issue has to be valid.
>
> Planning to reject the escalation and leave the issue as it is.

"if not it should have been made known as a known consideration", as said in my previous comment, this would be an obviously known issue to the team given witnetv2 is still rolling out. And how is this assigned a medium severity? I have submitted this same issue as an informational issue here https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/169 because at best it is just providing information which the sponsor would have realized before deployment anyways

WangSecurity

The reason why this is assigned medium severity is because the sponsor specifically asked if their contracts can be deployed on any chain as-is. This issue doesn't allow for such deployment. Hence, it's medium severity, cause the sponsor specifically asked for this.

The decision remains the same, planning to reject the escalation and leave the issue as it is.

10xhash

> The reason why this is assigned medium severity is because the sponsor specifically asked if their contracts can be deployed on any chain as-is. This issue doesn't allow for such deployment. Hence, it's medium severity, cause the sponsor specifically asked for this.
>
> The decision remains the same, planning to reject the escalation and leave the issue as it is.

If taking the context into the consideration (rolling out v2, outdated docs, no publicly available source of truth for which chains v2 is deployed etc.), it becomes clear that the sponsor is well aware of the state of witnet and is asking for specific chain related issues and not of the existence of witnet v2 in chains

It would be using sponsor's words too literally and if that is the case I would like to know if the issue "zerion chain is not live and hence contracts cannot be deployed" would also be a valid medium severity issue?

WangSecurity

I understand that it's obvious, but still the sponsor asked if the protocol can be deployed without any issues, and this issue shows that there would be. And it's valid because the protocol asked about it in README.

About Zerion, I would say this is invalid, because there would be nowhere to deploy, while in this case you can deploy the contracts, but they won't function correctly.

Hope that answers your question, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 10xhash: rejected

Issue M-14: `DrawManager.canStartDraw` does not consider retried RNG requests when determining if a new draw auction can be started

Source:
https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/129

Found by

0x73696d616f, berndartmueller, dany.armstrong90, hash, trachev, ydlee

Summary

Inconsistent checks in the `DrawManager.canStartDraw` function, neglecting to consider retried RNG requests, might lead to wrongly assuming that a new draw auction cannot be started.

Vulnerability Detail

The `DrawManager.canStartDraw` function checks if the `startDraw` function can be called. However, the checks are not consistent with the `startDraw` function. Specifically, the check in line `289` to determine if the draw has expired is different than the auction duration check in the `startDraw` function in lines 250-251. The latter uses the last RNG request's `closedAt` timestamp to determine the elapsed **auction** time, to consider any retried failed RNG requests, while the former checks if the **draw** has expired, not considering retried RNG requests.

As a result, if for a given draw a RNG request has been retried, and thus the total elapsed time from the draw close until now (`block.timestamp`) might exceed the auction duration, off-chain actors calling the `canStartDraw` function might wrongly assume that the draw auction can not be started, even though such a call would succeed.

Impact

As `canStartDraw` is also called internally by the startDrawReward function and both functions are likely to be used by off-chain actors to determine if a new draw auction an be started, this might lead to wrongly assuming that a new draw auction cannot be started, even though it should be possible. As a result, the current draw might not get awarded.

Code Snippet

DrawManager.canStartDraw()

```
275: /// @notice Checks if the start draw can be called.
276: /// @return True if start draw can be called, false otherwise
277: function canStartDraw() public view returns (bool) {
278:   uint24 drawId = prizePool.getDrawIdToAward();
279:   uint48 drawClosesAt = prizePool.drawClosesAt(drawId);
280:   StartDrawAuction memory lastStartDrawAuction = getLastStartDrawAuction();
281:   return (
282:     (
283:       // if we're on a new draw
284:       drawId != lastStartDrawAuction.drawId ||
285:       // OR we're on the same draw, but the request has failed and we
 ↪  haven't retried too many times
286:       (rng.isRequestFailed(lastStartDrawAuction.rngRequestId) &&
 ↪  _startDrawAuctions.length <= maxRetries)
287:     ) && // we haven't started it, or we have and the request has failed
288:     block.timestamp >= drawClosesAt && // the draw has closed
289:   _computeElapsedTime(drawClosesAt, block.timestamp) <= auctionDuration //
 ↪  the draw hasn't expired
290:   );
291: }
```

Tool used

Manual Review

Recommendation

Consider using the last request's `closedAt` timestamp instead of `drawClosesAt` to determine if the auction has expired to consider failed RNG requests that have been retried by calling `startDraw` again.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-draw-manager/pull/15

10xhash

Fixed. Now the lastStartDrawAuction.closedAt is used incase there are failed requests

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-15: User's might be able to claim their prizes even after shutdown

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/133

Found by

Rhaydden, hash

Summary

User's might be able to claim their prizes even after shutdown due to `lastObservationAt` and draw period difference

Vulnerability Detail

There is no shutdown check kept on the `claimPrize` function. Hence user's can calim their prizes even after the pool has been shutdown if the draw has not been finalized

link

```
function claimPrize(
  address _winner,
  uint8 _tier,
  uint32 _prizeIndex,
  address _prizeRecipient,
  uint96 _claimReward,
  address _claimRewardRecipient
) external returns (uint256) {
  /// @dev Claims cannot occur after a draw has been finalized (1 period after a
 ↪  draw closes). This prevents
  /// the reserve from changing while the following draw is being awarded.
  uint24 lastAwardedDrawId_ = _lastAwardedDrawId;
  if (isDrawFinalized(lastAwardedDrawId_)) {
    revert ClaimPeriodExpired();
  }
```

The remaining balance after a shutdown is supposed to be allocated to user's based on their (vault prize contribution + twab contribution) via the `withdrawShutdownBalance` and is not supposed to be based on a random number ie. via `claimPrize`

link

```
function withdrawShutdownBalance(address _vault, address _recipient) external
↪  returns (uint256) {
  if (!isShutdown()) {
    revert PrizePoolNotShutdown();
  }
```

In case the draw period is different from the TWAB's period length, it is not necessary that the shutdown due to the `lastObservationAt` occurs at the end of a draw. In such a case, it will allow user's who are winners of the draw to claim their prizes and also withdraw their share of the `shutdown balance` hence stealing funds from others

POC

```
diff --git a/pt-v5-prize-pool/test/PrizePool.t.sol
↪  b/pt-v5-prize-pool/test/PrizePool.t.sol
index 99fe6b5..5ce7ad6 100644
--- a/pt-v5-prize-pool/test/PrizePool.t.sol
+++ b/pt-v5-prize-pool/test/PrizePool.t.sol
@@ -75,6 +75,7 @@ contract PrizePoolTest is Test {
   uint256 RESERVE_SHARES = 10;

   uint24 grandPrizePeriodDraws = 365;
+  uint periodLength;
   uint48 drawPeriodSeconds = 1 days;
   uint24 drawTimeout; // = grandPrizePeriodDraws * drawPeriodSeconds; // 1000
↪  days;
   uint48 firstDrawOpensAt;
@@ -112,27 +113,26 @@ contract PrizePoolTest is Test {

   ConstructorParams params;

-  function setUp() public {
-    drawTimeout = 30; //grandPrizePeriodDraws;
-    vm.warp(startTimestamp);
+    function setUp() public {
+      // at end drawPeriod == 2 day, and period length in twab = 1 day

+    periodLength = 1 days;
+    drawPeriodSeconds = 2 days;
+
+    // the last draw should be ending at lastObservation timestamp + 1 day
+    startTimestamp = 1000 days;
```

```
+    firstDrawOpensAt =  uint48((type(uint32).max / periodLength ) % 2 == 0 ?
↪  startTimestamp + 1 days : startTimestamp + 2 days);
+
+
+
+    drawTimeout = 25854; // to avoid shutdown by drawTimeout when warping
+
+    vm.warp(startTimestamp + 1);
     prizeToken = new ERC20Mintable("PoolTogether POOL token", "POOL");
-    twabController = new TwabController(uint32(drawPeriodSeconds),
↪  uint32(startTimestamp - 1 days));
+    twabController = new TwabController(uint32(periodLength),
↪  uint32(startTimestamp));

-    firstDrawOpensAt = uint48(startTimestamp + 1 days); // set draw start 1 day
↪  into future
     initialNumberOfTiers = MINIMUM_NUMBER_OF_TIERS;

-    vm.mockCall(
-      address(twabController),
-      abi.encodeCall(twabController.PERIOD_OFFSET, ()),
-      abi.encode(firstDrawOpensAt)
-    );
-    vm.mockCall(
-      address(twabController),
-      abi.encodeCall(twabController.PERIOD_LENGTH, ()),
-      abi.encode(drawPeriodSeconds)
-    );
-
     drawManager = address(this);
     vault = address(this);
     vault2 = address(0x1234);
@@ -142,7 +142,7 @@ contract PrizePoolTest is Test {
       twabController,
       drawManager,
       tierLiquidityUtilizationRate,
-      drawPeriodSeconds,
+      uint48(drawPeriodSeconds),
       firstDrawOpensAt,
       grandPrizePeriodDraws,
       initialNumberOfTiers, // minimum number of tiers
@@ -155,6 +155,51 @@ contract PrizePoolTest is Test {
     prizePool = newPrizePool();
   }

+  function testHash_CanClaimPrizeAfterShutdown() public {
```

```
+    uint secondLastDrawStart = startTimestamp + (type(uint32).max /
↪  periodLength ) * periodLength - 3 days;
+
+    vm.warp(secondLastDrawStart + 1);
+    address user1 = address(100);
+    //address user2 = address(200);
+
+    // mint tokens to the user in twab
+    twabController.mint(user1, 10e18);
+    //twabController.mint(user2, 5e18);
+
+    // contribute prize tokens to the vault
+    prizeToken.mint(address(prizePool),100e18);
+    prizePool.contributePrizeTokens(address(this),100e18);
+
+    // move to the next draw and award this one
+    vm.warp(secondLastDrawStart + drawPeriodSeconds + 1);
+    prizePool.awardDraw(100);
+    uint drawId = prizePool.getOpenDrawId();
+
+    //currently not shutdown. but shutdown will occur in the middle of this
↪  draw allowing both prize claiming and the shutdown withdrawal
+    uint shutdownTimestamp = prizePool.shutdownAt();
+    assert(shutdownTimestamp == secondLastDrawStart + drawPeriodSeconds +
↪  periodLength);
+
+    vm.warp(shutdownTimestamp);
+
+    // call to store the shutdown data before the prize is claimed
+    prizePool.shutdownBalanceOf(address(this),user1);
+
+    /**
+     * address _winner,
+    uint8 _tier,
+    uint32 _prizeIndex,
+    address _prizeRecipient,
+    uint96 _claimReward,
+    address _claimRewardRecipient
+     */
+    prizePool.claimPrize(user1,1,0,user1,0,address(0));
+
+    // no withdrawing shutdown balance will revert due to the amount being
↪  withdrawn earlier via the claimPrize function
+    vm.prank(user1);
+    vm.expectRevert();
+    prizePool.withdrawShutdownBalance(address(this),user1);
+  }
```

```
+
    function testConstructor() public {
      assertEq(prizePool.firstDrawOpensAt(), firstDrawOpensAt);
      assertEq(prizePool.drawPeriodSeconds(), drawPeriodSeconds);
```

Impact

User's can double claim assets when vault shutdowns eventually

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L517-L524

Tool used

Manual Review

Recommendation

Add a notShutdown modifier to the claimPrize function

Discussion

0xjuaan

Escalate

This issue is informational

This issue requires the following to be true as stated by the watson

> In case the draw period is different from the TWAB's period length.....

The draw period and TWAB's period length are both set by the protocol themselves so setting this incorrectly will be admin error and considered invalid

By looking at the test suite we can clearly see that draw period = TWAB's period length == 1 day

sherlock-admin3

> Escalate
>
> This issue is informational
>
> This issue requires the following to be true as stated by the watson

> > In case the draw period is different from the TWAB's period length.....
>
> The draw period and TWAB's period length are both set by the protocol themselves so setting this incorrectly will be admin error and considered invalid
>
> By looking at the test suite we can clearly see that draw period = TWAB's period length == 1 day

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

10xhash

Escalate

The issue allows the winners to claim the rewards twice which causes other user's to loose their prizes which I think should warrant high severity

sherlock-admin3

> Escalate
>
> The issue allows the winners to claim the rewards twice which causes other user's to loose their prizes which I think should warrant high severity

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

@trmid @10xhash Might want to take a look at this comment. If not I rated it medium given the unlikely likelihood of shutdown mentioned in comments here.

0x73696d616f

I believe for this to happen, 2 things must be considered:

- The draw length must be different than the twab length.
- The shutdown is caused by the end of the twab controller, in 82 years.

Given these extensive limitations, medium is appropriate. If the shutdown happens due to the rng service, it ends at the end of the draw so prizes can not be claimed and this issue does not exist.

10xhash

> @trmid @10xhash Might want to take a look at this comment. If not I rated it medium given the unlikely likelihood of shutdown mentioned in comments here.

It is a not an incorrect setting. The requirements for the draw period are that:

```
if (
  params.drawPeriodSeconds < twabPeriodLength ||
  params.drawPeriodSeconds % twabPeriodLength != 0
) {
  revert IncompatibleTwabPeriodLength();
}
```

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-prize-pool/src/PrizePool.sol#L357-L362

In the sponsors comment, it is mentioned that the case of `the TWAB timestamps have reached their max limit` is being explicitly considered which is the scenario discussed here

```
The second is guaranteed to happen at the end of life for a TWAB controller and
↪  was a major consideration in the design of the shutdown logic
```

WangSecurity

Since there's a set of inputs that doesn't result in an issue, we should consider this set will be used by the protocol team. Planning to accept the escalation and invalidate the issue, cause as I understand there won't be an issue when TWAB and draw periods are the same.

10xhash

> Since there's a set of inputs that doesn't result in an issue, we should consider this set will be used by the protocol team. Planning to accept the escalation and invalidate the issue, cause as I understand there won't be an issue when TWAB and draw periods are the same.

How is it considered that the protocol will be using drawPeriod == TWABperiod when the code clearly has other rules to determine whether a drawPeriod is compatible or not?

```
if (
  params.drawPeriodSeconds < twabPeriodLength ||
```


```
    params.drawPeriodSeconds % twabPeriodLength != 0
  ) {
    revert IncompatibleTwabPeriodLength();
  }
```

WangSecurity

If there are values set by admins and only specific set causes issues, then it's considered an admin mistake to set values to cause issues. But, looking at this issue and the code again, I agree it wouldn't be a mistake, cause draw period can be any multiple of TWAB period length. Hence, I agree this scenario is indeed possible.

Planning to reject the escalation and leave the issue as it is.

10xhash

> If there are values set by admins and only specific set causes issues, then it's considered an admin mistake to set values to cause issues. But, looking at this issue and the code again, I agree it wouldn't be a mistake, cause draw period can be any multiple of TWAB period length. Hence, I agree this scenario is indeed possible.
>
> Planning to reject the escalation and leave the issue as it is.

I had raised an escalation to reconsider the severity of the issue as high which I don't think have been addressed/considered. Can you please look into it

WangSecurity

In the escalation message you say the user is able to claim the prize twice, you mean claim the prize before the shutdown and after the shutdown if the current draw haven't been finalised. Or by that you mean claim the prize and withdraw their share of the shutdown balance?

10xhash

> mean claim the prize and withdraw their share of the shutdown balance?

"claim the prize and withdraw their share of the shutdown balance?" this. Sorry for the incorrectness there

WangSecurity

Then I agree it should be high. Yes, only the winners can do it, but any winner can do it without any extensive limitations. Planning to reject @0xjuaan escalation since the issue should remain valid. Planning to accept @10xhash's escalation and upgrade severity to high.

0x73696d616f

@WangSecurity what are your thoughts on these 2 points? The parameter being a specific one and the issue happening in 82 years are extensive limitations (we will never witness it, or there is a 0.0000001% chance). Agree that the sponsor saying they care about the shutdown makes the issue in scope, but high for a issue that will not happen seems too much.

WangSecurity

Thank you and excuse me, TWAB limit is always 82 years, correct? And if the shutdown happens due to draw timeout has been reached, then this issue won't happen?

0x73696d616f

@WangSecurity I think it does not happen due to the draw timeout, look at the following links to confirm.

Shutdown check

`shutdownAt()` returns drawTimeoutAt

drawTimeoutAt

drawClosesAt

On the other hand, the twab controller timeout, lastObservationAt(), returns approx 82 years in the future (type(uint32).max).

WangSecurity

And if it can't happen at the draw timeout, because in that case the last draw would have been finalised, and this issue requires the opposite, correct?

Moreover, is it guaranteed that when TWAB limit is reached, there would be a not finalised draw, or it depends on configuration?

0x73696d616f

> And if it can't happen at the draw timeout, because in that case the last draw would have been finalised, and this issue requires the opposite, correct?

Yes

> Moreover, is it guaranteed that when TWAB limit is reached, there would be a not finalised draw, or it depends on configuration?

Depends on config, the period length of the twab controller needs to be different than that of the prize pool for them to end at different timestamps and cause this issue

WangSecurity

Thank you for this clarifications, then I would agree this is a high constraint, and even though any winner in this case can steal funds, the TWAB period length of 82 years has to be reached, which may never happen due to draw timeout shutdown. Hence, I agree indeed medium severity should remain.

Planning to reject both escalation and leave the issue as it is.

10xhash

> Thank you for this clarifications, then I would agree this is a high constraint, and even though any winner in this case can steal funds, the TWAB period length of 82 years has to be reached, which may never happen due to draw timeout shutdown. Hence, I agree indeed medium severity should remain.
>
> Planning to reject both escalation and leave the issue as it is.

Here the project is especially considering this scenario of the max time reaching ie.type(uint32).max. And the other constraint of draw period length being not equal to the twab period length is also not a constraint as such. Both of these are values/situations that are supposed to be handled by the contract

WangSecurity

I agree that the draw period being not equal to the TWAB period length is not an extensive constraint. But I believe reaching the TWAB period length of 82 years without a draw timeout, is quite a high constraint. Moreover, this scenario may never occur. Hence, I believe medium is more appropriate.

The decision remains the same, reject both escalations and leave the issue as it is.

10xhash

> I agree that the draw period being not equal to the TWAB period length is not an extensive constraint. But I believe reaching the TWAB period length of 82 years without a draw timeout, is quite a high constraint. Moreover, this scenario may never occur. Hence, I believe medium is more appropriate.
>
> The decision remains the same, reject both escalations and leave the issue as it is.

A draw timeout would occur when the protocol is non-operational ie. the bots refuse to award draws. The team wants to consider this project to be work correctly even after 82 years. I am reiterating the comment of the sponsor:

```
The second is guaranteed to happen at the end of life for a TWAB controller and
↪  was a major consideration in the design of the shutdown logic
```

It is different if the project simply decides to not handle the scenario after type(uint32).max and if the project explicitly wants to handle the scenario after this time period. It is upto the team to decide which timeframe they want to provide support for and if they want the timeframe to be even 1000 years, it shouldn't be considered as a limitation.

WangSecurity

I understand that this is the scenario the protocol wants to cover, but this doesn't change the fact that 82 years is quite long and there are others reasons why the vault may shutdown before this 82 years expire. Hence, I believe the medium is more appropriate. I don't mean to say this is invalid, or not of importance, but I see it as an extensive limitation, because of which this issue may never arise.

Hence, the decision remains the same, planning to reject both escalation and leave the issue medium as it is.

10xhash

From sherlocks docs, criteria for high:

```
Definite loss of funds without (extensive) limitations of external conditions
```

If the protocol plans to operate past 82 years, is that timeframe still considered under extensive limitation of external condition?

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-prize-pool/pull/116

WangSecurity

I see that the protocol plans to operate for 82 years, but given the fact, that it's a very long timeframe and other reasons can cause shutdown before it and won't cause this issue, medium is more appropriate. It's not based only on the fact that the TWAB limit is 82 years, but on the combination of factors.

The decision remains the same, planning to reject both escalation and leave the issue as it is.

10xhash

> I see that the protocol plans to operate for 82 years, but given the fact, that it's a very long timeframe and other reasons can cause shutdown before it and won't cause this issue, medium is more appropriate. It's not based only on the fact that the TWAB limit is 82 years, but on the combination of factors.

> The decision remains the same, planning to reject both escalation and leave the issue as it is.

There is one other reason due to which the protocol can shutdown and that is if the draw is not awarded for a time period of drawTimeOut. If the protocol is assumed to operate 82 yrs, isn't it trivial that it is also assumed that the bots will continue to draw awards during this period

WangSecurity

I still believe that medium severity is more appropriate here, based on the same reasons as here.

The decision remains the same, planning to reject both escalations and leave the issue as it is.

10xhash

> I still believe that medium severity is more appropriate here, based on the same reasons as here.
>
> The decision remains the same, planning to reject both escalations and leave the issue as it is.

ok. I have no more points to make

WangSecurity

Result: Medium Has duplicates

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- 0xjuaan: rejected
- 10xhash: rejected

10xhash

Fixed Now the shutdownTime is moved to the start of the corresponding draw

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-16: `maxDeposit` doesn't comply with ERC-4626

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/134

Found by

0x73696d616f, Tri-pathi, hash

Summary

`maxDeposit` doesn't comply with ERC-4626 since depositing the returned amount can cause reverts

Vulnerability Detail

The contract's `maxDeposit` function doesn't comply with ERC-4626 which is a mentioned requirement. According to the specification, `MUST return the maximum amount of assets deposit would allow to be deposited for receiver and not cause a revert ....`
The `deposit` function will revert in case the deposit is a lossy deposit ie. totalPreciseAsset function returns less than the totalDebt after the deposit. It is possible for this to occur due to rounding inside the preview redeem function of the yieldVault in the absence / depletion of yield buffer

POC

Add the following test inside `pt-v5-vault/test/unit/PrizeVault/PrizeVault.t.sol`

```
function testHash_MaxDepositRevertLossy() public {
    PrizeVault testVault=  new PrizeVault(
        "PoolTogether aEthDAI Prize Token (PTaEthDAI)",
        "PTaEthDAI",
        yieldVault,
        PrizePool(address(prizePool)),
        claimer,
        address(this),
        YIELD_FEE_PERCENTAGE,
        0,
        address(this)
    );

    underlyingAsset.mint(address(yieldVault),100e18);
    YieldVault(address(yieldVault)).mint(address(100),99e18); // 99 shares , 100
↪   assets
```

```
    uint maxDepositReturned = testVault.maxDeposit(address(this));

    uint amount_ = 99e18;
    underlyingAsset.mint(address(this),amount_);
    underlyingAsset.approve(address(testVault),amount_);

    assert(maxDepositReturned > amount_);

    vm.expectRevert();
    testVault.deposit(amount_,address(this));

}
```

Impact

Failure to comply with the specification which is a mentioned necessity

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b659585e4c7a6b9b652fb075f86db3/pt-v5-vault/src/PrizeVault.sol#L991-L992

Tool used

Manual Review

Recommendation

Consider the yieldBuffer balance too inside the `maxDeposit` function

Discussion

sherlock-admin4

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> deposit can only incur rounding issue if yield buffer is depleted but if the buffer is depleted reverts on deposit are expected__ see L112-114 of PrizeVault.sol

nevillehuang

Valid medium since it was mentioned as:

> Is the codebase expected to comply with any EIPs? Can there be/are there any deviations from the specification? PrizeVaults are expected to strictly comply with the ERC4626 standard.

Sherlock rules states

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-vault/pull/113

10xhash

Fixed. Now maxDeposit returns 0 unless totalAssets is >= totalDebt + yieldBuffer/2. This will ensure that unless rounding error amounts to yieldBuffer/2, lossyDeposit will not occur

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-17: `maxRedeem` doesn't comply with ERC-4626

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/136

Found by

hash

Summary

`maxRedeem` function reverts due to division by 0 and hence doesn't comply with ERC4626

Vulnerability Detail

The contract's `maxRedeem` function doesn't comply with ERC-4626 which is a mentioned requirement. According to the specification, `MUST NOT revert.`
The `maxRedeem` function will revert in case the totalAsset amount is 0 due to a divison by 0

link

```
uint256 _maxScaledRedeem = _convertToShares(_maxWithdraw, _totalAssets,
↪  totalDebt(), Math.Rounding.Up);
```

link

```
function _convertToShares(
    uint256 _assets,
    uint256 _totalAssets,
    uint256 _totalDebt,
    Math.Rounding _rounding
) internal pure returns (uint256) {
    if (_totalAssets >= _totalDebt) {
        return _assets;
    } else {
        return _assets.mulDiv(_totalDebt, _totalAssets, _rounding);
```

POC

```
diff --git a/pt-v5-vault/test/unit/PrizeVault/PrizeVault.t.sol
↪  b/pt-v5-vault/test/unit/PrizeVault/PrizeVault.t.sol
index 7006f44..9b92c35 100644
```

```
--- a/pt-v5-vault/test/unit/PrizeVault/PrizeVault.t.sol
+++ b/pt-v5-vault/test/unit/PrizeVault/PrizeVault.t.sol
@@ -4,6 +4,8 @@ pragma solidity ^0.8.24;
 import { UnitBaseSetup, PrizePool, TwabController, ERC20, IERC20, IERC4626,
 ↪  YieldVault } from "./UnitBaseSetup.t.sol";
 import { IPrizeHooks, PrizeHooks } from
 ↪  "../../../src/interfaces/IPrizeHooks.sol";
 import { ERC20BrokenDecimalMock } from
 ↪  "../../contracts/mock/ERC20BrokenDecimalMock.sol";
+import "forge-std/Test.sol";
+

 import "../../../src/PrizeVault.sol";

@@ -45,6 +47,34 @@ contract PrizeVaultTest is UnitBaseSetup {
         assertEq(testVault.owner(), address(this));
     }

+    function testHash_MaxRedeemRevertDueToDivByZero() public {
+        PrizeVault testVault=  new PrizeVault(
+            "PoolTogether aEthDAI Prize Token (PTaEthDAI)",
+            "PTaEthDAI",
+            yieldVault,
+            PrizePool(address(prizePool)),
+            claimer,
+            address(this),
+            YIELD_FEE_PERCENTAGE,
+            0,
+            address(this)
+        );
+
+        uint amount_ = 1e18;
+        underlyingAsset.mint(address(this),amount_);
+
+        underlyingAsset.approve(address(testVault),amount_);
+
+        testVault.deposit(amount_,address(this));
+
+        // lost the entire deposit amount
+        underlyingAsset.burn(address(yieldVault), 1e18);
+
+        vm.expectRevert();
+        testVault.maxRedeem(address(this));
+
+    }
+
     function testConstructorYieldVaultZero() external {
```

```
        vm.expectRevert(abi.encodeWithSelector(PrizeVault.YieldVaultZeroAddress
↪   .selector));
```

Impact

Failure to comply with the specification which is a mentioned necessity

Code Snippet

Tool used

Manual Review

Recommendation

Handle the `_totalAssets == 0` condition

Discussion

sherlock-admin3

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> if `totalAssets` is 0 the first if statement will be executed thus no revert

nevillehuang

Valid medium since it was mentioned as:

> Is the codebase expected to comply with any EIPs? Can there be/are there any deviations from the specification? PrizeVaults are expected to strictly comply with the ERC4626 standard.

Sherlock rules states

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity

InfectedIsm

@nevillehuang @WangSecurity I think you didn't take my judging comment into consideration, sorry for acting so late.

If `totalAssets == 0`, this means `totalDebt == 0` too as:

- at start, the prize vault gets assets deposited as the yield buffer so `totalAssets > 0`
- debt is minted 1:1 with deposited assets, so `TotalAssets > TotalDebt` during the existence of the vault
- when yield is accrued, TotalAssets is increased, while debt is as most increased up to totalAssets for yield to be liquidated
- when the Yield Vault take a loss for any reason, then assets<debt, a user withdrawing at this moment `shares (debt) burned > asset withdrawn`, until `debt = assets`
- This means, if `totalAssets == 0`, then `totalDebt == 0` too

Hence, this is the case `if (_totalAssets >= _totalDebt)` that is executed, and not the `else` case, so no division by 0 here.

The only case where this could happen is a yield vault being hacked and fully withdrawn from its asset, and the yield buffer of the prize vault fully emptied If this cannot happen in normal circonstances (yield vault compliant, so no way to get hacked) then we should consider this case cannot happen, so the function cannot revert.

@10xhash FYI

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-vault/pull/112

10xhash

Fixed Now maxRedeem returns 0 instead of reverting

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-18: Users can setup hooks to control the expansion of tiers

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/162

The protocol has acknowledged this issue.

Found by

hash

Summary

Reentrancy allows to not claim rewards and update tiers

Vulnerability Detail

The fix for the issue still allows user's claim the rewards. But as mitigation, the contract will revert. But this still allows the user's to specifically revert on just the last canary tiers in order to always prevent the expansion of the tiers

Impact

User's have control over the expansion of tiers

Code Snippet

https://github.com/sherlock-audit/2024-05-pooltogether/blob/1aa1b8c028b655985e4c7a6b9b652fb075f86db3/pt-v5-vault/src/abstract/Claimable.sol#L87

Tool used

Manual Review

Recommendation

Discussion

sherlock-admin4

1 comment(s) were left on this issue during the judging contest.

infect3d commented:

> expected design

nevillehuang

request poc

sherlock-admin4

PoC requested from @10xhash

Requests remaining: **8**

10xhash

The issue that I wanted to point out is that since a user is not going to be benefited if a canary tier prize is claimed (the claiming bot would keep the fee as the entire prize amount. The canary prize claiming is kept inorder to determine if the number of tiers should be increased / decreased) a user can revert in the hook if the tier is a canary tier. If it is misunderstood as a single user can disallow the entire tier expansion, it is not so

```
diff --git a/pt-v5-vault/test/unit/Claimable.t.sol
↪  b/pt-v5-vault/test/unit/Claimable.t.sol
index dfbb351..44b6568 100644
--- a/pt-v5-vault/test/unit/Claimable.t.sol
+++ b/pt-v5-vault/test/unit/Claimable.t.sol
@@ -131,6 +131,22 @@ contract ClaimableTest is Test, IPrizeHooks {
         assertEq(logs.length, 1);
     }

+
+
+    function testHash_RevertIfCanaryTiers() public {
+        PrizeHooks memory beforeHookOnly = PrizeHooks(true, false, hooks);
+
+        vm.startPrank(alice);
+        claimable.setHooks(beforeHookOnly);
+        vm.stopPrank();
+
+        mockClaimPrize(alice, 1, 2, prizeRedirectionAddress, 1e17, bob);
+
+        vm.expectRevert();
+
+        claimable.claimPrize(alice, 1, 2, 1e17, bob);
+    }
+
     function testClaimPrize_afterClaimPrizeHook() public {
         PrizeHooks memory afterHookOnly = PrizeHooks(false, true, hooks);

@@ -259,6 +275,13 @@ contract ClaimableTest is Test, IPrizeHooks {
         uint96 reward,
```

```
          address rewardRecipient
      ) external returns (address, bytes memory) {
+         // prizePool.isCanarayTier(tier)
+         bool isCanaryTier = true;
+
+         if(isCanaryTier){
+             revert();
+         }
+
          if (useTooMuchGasBefore) {
              for (uint i = 0; i < 1000; i++) {
                  someBytes = keccak256(abi.encode(someBytes));
```

trmid

Coincidentally, about 2 weeks ago I started writing and sharing an article about how depositors can use this as a way to "vote" on prize sizes. Recently published the article showing how this can be used to democratize the prize algorithm.

> User's have control over the expansion of tiers

This is permissionless and powerful!

One dev's bug is another dev's feature.

0xjuaan

Escalate

The following is the criteria for medium severity issues

1. Causes a loss of funds but requires certain external conditions or specific states, or a loss is highly constrained. The losses must exceed small, finite amount of funds, and any amount relevant based on the precision or significance of the loss.
2. Breaks core contract functionality, rendering the contract useless or leading to loss of funds.

This issue is informational

The impact of this issue as described by the watson is

> User's have control over the expansion of tiers

This does not lead to loss of funds AND does not break core contract functionality. Therefore the issue does not satisfy sherlock's requirements for medium severity

sherlock-admin3

Escalate

The following is the criteria for medium severity issues

1. Causes a loss of funds but requires certain external conditions or specific states, or a loss is highly constrained. The losses must exceed small, finite amount of funds, and any amount relevant based on the precision or significance of the loss.
2. Breaks core contract functionality, rendering the contract useless or leading to loss of funds.

This issue is informational

The impact of this issue as described by the watson is

> User's have control over the expansion of tiers

This does not lead to loss of funds AND does not break core contract functionality. Therefore the issue does not satisfy sherlock's requirements for medium severity

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

@10xhash What is the impact of your PoC mentioned here?

10xhash

> @10xhash What is the impact of your PoC mentioned here?

User's can revert specifically on prize claims occurring for canary tiers. The expansion of tiers (number of tiers determine the size and the number of prizes) is based on whether the canary tier prizes are claimed or not and hence this allows user's to influence it without any loss to them (since they gain nothing from a canary tier claim)

WangSecurity

To clarify, the attack is pretty much the same as for the issue from C4 audit, but here it doesn't allow to bypass the fees but to prevent the expansion of tiers?

And about the impact, this allows users to increase their chance of winning, or just the size and number of prizes are smaller (i.e. what is the benefit for the attacker to control expansion of tiers or loss to others?).

10xhash

> To clarify, the attack is pretty much the same as for the issue from C4 audit, but here it doesn't allow to bypass the fees but to prevent the expansion of tiers?
>
> And about the impact, this allows users to increase their chance of winning, or just the size and number of prizes are smaller (i.e. what is the benefit for the attacker to control expansion of tiers or loss to others?).

Yes

It doesn't improve a user's chance of winning directly. If some user's want to have a specific distribution of prizes (eg: instead of distributing prizes more frequently, they want to have a large yearly prize), they can influence to achieve this

Hash01011122

Attack Vector: But as mitigation, the contract will revert. But this still allows the user's to specifically revert on just the last canary tiers in order to always prevent the expansion of the tiers

Impact: User's have control over the expansion of tiers

As pointed out by @0xjuaan this issue has no impact on protocol whatsoever. This should be considered as low/info issue.

Another point to note from the sponsors comment:

> One dev's bug is another dev's feature.

This is a design choice of protocol and can be invalidated.

@nevillehuang @WangSecurity Do correct me if I am wrong

WangSecurity

Firstly, as I understand, it's not a design decision, but an acceptable risk, which was not documented (this is confirmed). Secondly, I believe it allows the users to control tiers to their needs, an examples is given by the submitter here. Since, it wasn't documented as acceptable, and gives users more control than they should have, which I believe breaks core contract functionality. But if it's incorrect please correct me.

Planning to reject the escalation and leave the issue as it is.

Hash01011122

@WangSecurity I guess you are getting the escalation wrong, even if we overlook the fact of design choice then there isn't any direct impact nor breakage of core functionality.

> If some user's want to have a specific distribution of prizes (eg: instead of distributing prizes more frequently, they want to have a large yearly

prize), they can influence to achieve this.

Users can influence it in some cases to benefit from it increasing their odds, this doesn't qualify as direct impact.

This is low/info severity issue.

WangSecurity

As I understand it indeed has impact, the user is able to control distribution of prizes (i.e. get a large yearly prize instead of frequent prizes), when they shouldn't be able to. And it wasn't the design, hence, I see it as a valid issue.

The decision remains the same, planning to reject the escalation and leave the issue as it is.

Hash01011122

@10xhash @trmid What large yearly prize instead of frequent prizes will account to? In terms of loss incurred? Also given the timeframe wouldn't protocol notice it and can redeploy tier contracts? And why this isn't design choice and acceptable risk @WangSecurity, because codebase is clearly designed the way @trmid pointed out.

WangSecurity

I still believe that this is a vulnerability because the users gain control that they shouldn't have and can prevent the expansion of tiers. It will have rather a long-term effect and can lead to this user getting more wins than they should've if they hadn't this control.

I can confirm it's not a design decision and it's still the same issue as it was in the previous audit, but the impact is lower. It's not an acceptable risk because it wasn't put in a special field in the README and given the fact that the sponsors view it as good, rather than bad, doesn't mean we have to straight invalidate it.

The decision remains the same, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 0xjuaan: rejected

Issue M-19: Gas Manipulation by Malicious Winners in claimPrizes Function

Source: https://github.com/sherlock-audit/2024-05-pooltogether-judging/issues/163

Found by

0x73696d616f, 0xSpearmint1, MiloTruck, infect3d, jo13

Summary

A malicious winner can exploit the `claimPrizes` function in the `Claimer` contract by reverting the transaction through returning a huge data chunk. This manipulation can cause the transaction to run out of gas, preventing legitimate claims and allowing the malicious user to claim prizes without computing winners.

Vulnerability Detail

- The `Claimer` contract allows users to claim prizes on behalf of others by calling the `claimPrizes` function.
- A malicious winner can exploit this function by returning a huge data chunk when called, causing the transaction's gas to be too high and revert.
- Although the function catches the revert, the remaining gas (63/64 of the original gas) is likely insufficient for the rest of the claims.
- The malicious winner can then replay the transaction to claim the fees from the first claimer's computation without needing to compute the winners themselves.

Impact

- Legitimate claimers may lose gas fees due to transaction reverts caused by malicious winners.
- Malicious winners can exploit this to claim prizes without computing winners, undermining the fairness of the prize distribution.

Recommendation

- Implement a gas limit check to ensure that sufficient gas remains for the rest of the claims after catching a revert.

- Consider adding a mechanism to penalize or blacklist addresses that attempt to exploit this vulnerability.

Discussion

InfectedIsm

Escalate

issue #110 doesn't talk about gas bomb through return data, and shouldn't be a duplicate of this issue. If there's an issue that could be seen as similar, it would be #73, as in that one the attacker steal the reward that was attributed to the original claimer

sherlock-admin3

> Escalate
>
> issue #110 doesn't talk about gas bomb through return data, and shouldn't be a duplicate of this issue. If there's an issue that could be seen as similar, it would be #73, as in that one the attacker steal the reward that was attributed to the original claimer

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

Escalation should be rejected, watson didn't pay attention to the correct duplication. #110 is already duplicated to #73

WangSecurity

Agree with the Lead Judge, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- InfectedIsm: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/GenerationSoftware/pt-v5-vault/pull/115

10xhash

Fixed Now ExcessivelySafeCall is used restricting the return/revert data copying to 128 bytes

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.