PoolTogether v3 Audit

OPENZEPPELIN SECURITY | OCTOBER 21, 2020

Security Audits

Get the title[uncode_info_box items=”Date,Categories,Author|no_avatar|inline_avatar|display_prefix” text_font=”font-202125″ text_transform=”uppercase” separator=”pipe”]

PoolTogether is a protocol that allows users to join a trust-minimized no-loss savings game on the Ethereum network.

We have previously audited the original system as well as the pods feature. The PoolTogether team asked us to review the `PrizePool` component of version 3 of the system. We looked at the code and now publish our results.

The audited commit is `23b826b94e62d9c3a49bbe637da44cd1409c5b7f`. The contracts and directories included in the scope were:

- contracts/prize-pool
- contracts/token
- contracts/utils/MappedSinglyLinkedList.sol

The rest of the system, notably the Prize Strategy contracts and the game mechanics, was not included in the scope. All external code and contract dependencies were assumed to work as documented.

Update: *The Pooltogether team fixed in individual pull requests the issues we reported. We refer to these updates in their corresponding issues. Our analysis of the mitigations assumes the pull*

Overall, we are happy with the security posture of the team and the health of the code base. As with both previous audits, we appreciate the small, encapsulated and well documented functions. We are also happy to see further generalizations of the design, which should assist with extensibility.

System Overview

Our original audit report contains a description of the system's overall functionality. However, the audited code includes a complete rewrite of the code base, so we will describe the implementation from first principles.

The core component is the `PrizePool` contract, which allows users to deposit funds, in the form of an ERC20 token, into the system. These funds are transferred to a third party yield service that generates interest. In particular, we reviewed how this contract is specialized for use with Compound Finance. In exchange, users are issued tokens that represent their claim on their deposits. In the interest of extensibility, the system supports any number of different tokens to represent the different possible claims. Initially, they will include tickets that are eligible to win awards in the savings game and sponsorship tokens that earn interest but are not eligible for awards.

Value transfers are mediated by a `PrizeStrategy` contract, which was not reviewed in this audit. Whenever funds are deposited or withdrawn, the prize strategy is informed so it can update the savings game. It may also apply an exit fee or timelock for withdrawals to discourage short-duration deposits. Both mechanisms are governed by the `PrizeStrategy` contract, but enforced by the `PrizePool` contract, which puts a hard limit on the severity of the penalties to prevent malicious values. Lastly, the `PrizeStrategy` implements the actual savings game and awards tokens (which may include external ERC20 and ERC721 tokens) accordingly.

The purpose of this design is to allow the `PrizePool`, which is mostly autonomous, to handle funding and token transfers, and to ensure users are able to withdraw their deposits. The `PrizeStrategy` contract is upgradeable and runs the actual savings game, but is only able to

Privileged Roles

The `PrizePool` contract is designed to minimize the admin privileges, but there are still some that should be recognized. When the contract is deployed, it is configured with a `PrizeStrategy` contract, a maximum exit fee rate, and a maximum withdrawal timelock duration. These values cannot be updated. However, the `PrizeStrategy` implements hooks that are called on all operations, and could potentially veto legitimate deposits and withdrawals. To address this, the `PrizePool` contract has an owner that can remove the dependence on the `PrizeStrategy`. This effectively triggers an emergency shutdown, where new deposits and awards stop functioning, but all users can withdraw their funds.

The contract owner can also add new tokens to represent different types of claims that users have on the funds. It should be noted that in all operations, users choose which of the available tokens to exchange for their deposits. In other words, users can choose to receive tickets or sponsorship tokens (or potentially new tokens that the owner adds), but they cannot be forced to exchange between them, so the ability to add new tokens can only increase the options.

It should also be noted that the `PrizePool` contract is designed to be GSN compliant, which means there is a trusted forwarder role that can send arbitrary transactions on behalf of any other user. Naturally, this is an extremely powerful role that should only be assigned to a well-restricted smart contract.

Ecosystem Dependencies

As the ecosystem becomes more interconnected, understanding the external dependencies and assumptions has become an increasingly crucial component of system security. To this end, we would like to discuss how the prize pool depends on the surrounding ecosystem.

Most importantly, all user deposits are immediately transferred to a third-party yield service (like Compound). If the service is low on liquidity (or otherwise behaving unexpectedly), users may not be able to recover their deposits. The system should only be used if the yield service is appropriately trusted.

a race to get a transaction confirmed before the deadline, so there is no additional risk.

Here we present our findings.

Critical

[C01] Funds can be lost

The `sweepTimelockBalances` function accepts a list of users with unlocked balances to distribute. However, if there are duplicate users in the list, their balances will be counted multiple times when calculating the total amount to withdraw from the yield service. This has two consequences:

- After the transaction is complete, the excess amount withdrawn will be held by the `PrizePool` contract (instead of the yield service) and will not earn interest
- Eventually, a user will want to withdraw that amount, which will fail when the `PrizePool` attempts to redeem it from the yield service. This means the last users to withdraw will lose their funds. Interestingly, in the case of the `CompoundPrizePool`, this is partially mitigated by the **"[H01] Improper Error Handling"** issue.

Consider checking for duplicate users when calculating the amount to withdraw.

***Update**: Fixed in pull request #100. The `_sweepTimelockBalances` function now removes the user's timelock balance after adding it to the total withdrawal amount and saving it into an auxiliary array named `balances`, which has the same size as the `users` array provided as a parameter. In this process, as the code still allows the `users` to have duplicate addresses, a zero will be saved in the `balances` array for each repeated address.*

High

[H01] Improper Error Handling

The `CompoundPrizePool` contract interacts with the Compound system to mint new cTokens and redeem them for the underlying asset. In both cases the code ignores the returned error code. Since these calls are intended to be part of atomic operations that mint and burn pool tokens, Consider reverting the transaction whenever the Compound system returns an error, to ensure the operations remain atomic.

***Update:** Fixed in pull request #101. The `_redeem` and `_supply` functions in `CompoundPrizePool.sol` now require that the value returned by the `cToken` contract equals zero, which represents the NO_ERROR code in Compound.*

[H02] Trapped sponsorship

The `withdrawInstantlyFrom` function of the `PrizePool` allows the caller to optionally provide a `sponsorAmount` that will be applied towards the exit fee. In practice, this should be equivalent to a simple transfer from the caller to the `from` address. Whether or not a `sponsorAmount` is provided, the exit fee should be implicitly levied by leaving this amount in the yield service. However, the prize pool only leaves the non-sponsored component of the exit fee in the yield service, which is the same amount that is transferred to the user. This means the sponsored amount remains trapped in the `PrizePool` contract and is not included in the next award. Consider adjusting the `_redeem` parameters so the full exit fee is left in the yield service.

***Update:** Fixed in pull request #102. The Pooltogether team decided to remove the sponsor amount from the `withdrawInstantlyFrom` function.*

Medium

[M01] Inconsistent list initialization

The `MappedSinglyLinkedList` library has an `initialize` function that accepts multiple addresses. However, it does not perform any validation of the user-supplied addresses. If the list contains duplicate addresses or the special `SENTINAL` address, it will be created in an inconsistent state. Consider using the `addAddress` function in the loop to perform the appropriate checks.

***Update:** Fixed in pull request #103. The `initialize` function no longer accepts an array of addresses as a parameter, and the `addAddresses` function was implemented to add multiple items to the list, handling both duplicate addresses and the `SENTINAL` address.*

The `withdrawInstantlyFrom` function of the `PrizePool` contract caps the exit fee with the `limitExitFee` function, and then immediately performs the same operation to limit it again. Consider removing the redundant code.

Update: *Fixed in pull request #104.*

[L02] Linked list contains `SENTINAL`

The `contains` function of the `MappedSinglyLinkedList` library will incorrectly return `true` when called with the `SENTINAL` address. Consider checking and returning `false` in this edge case.

Update: *Fixed in pull request #105. The `contains` function now returns false when called with the `SENTINAL` address.*

[L03] Lists can be initialized multiple times

The `MappedSinglyLinkedList` data structure can be initialized multiple times by calling any of the initialize functions, which can lead to several inconsistencies:

- Addresses in the original list cannot be added to the active one
- Addresses in the original list **can** be removed, despite not affecting the active list, which may lead to a negative integer overflow
- Addresses in the original list will appear to be contained in the active one

This does not occur in the audited code. Nevertheless, consider preventing initialization of non-empty lists.

Update: *Fixed in pull request #106. Now, the `initialize` function explicitly checks that the number of items in the list is zero before initialization.*

[L04] Lack of event emission after sensitive changes

The `addControlledToken` function of the `PrizePool` contract adds a new token to the `_tokens` array without emitting an event. To facilitate off-chain services tracking important state

***Update**: Fixed in pull request #107. The `addControlledToken` function now emits the `ControlledTokenAdded` event.*

[L05] `MappedSinglyLinkedList` Encapsulation

To improve encapsulation of the `MappedSinglyLinkedList` data structure, consider renaming both instances of the `currentToken` variable to `currentAddress`.

Additionally, consider including `start`, `end` and `next` functions so functions that traverse the list do not need to know the `addressMap` or `SENTINAL` implementation details.

***Update:** Fixed in pull request #108.*

[L06] Naming suggestions

The `PrizePool` contract implements the naming convention of prefixing `internal` values with an underscore. However, the following functions and variables disregard this convention:

- timelockBalances
- unlockTimestamps
- limitExitFee
- isControlled

Additionally, in the `MappedSinglyLinkedList` library, `SENTINAL` should be `SENTINEL`.

***Update:** Fixed in pull request #108 and pull request #109.*

Notes

[N01] Misleading comments

- The Ethereum Natural Specification comment on the `_redeem` function of the `CompoundPrizePool` contract incorrectly claims the parameter is the amount of yield-bearing tokens to redeem. In fact, it is the amount of the underlying asset token to retrieve.
- The Ethereum Natural Specification comment on the `withdrawWithTimelockFrom` function of the `PrizePool` contract states the unlock timestamp is the time *after which* the

***Update:** Fixed in pull request #110.*

[N02] Assigning zero address

In the `removeAddress` function of the `MappedSinglyLinkedList` library, the target address is cleared by assigning the zero address to a record in the `addressMap`. For clarity and consistency, consider deleting the record, as implemented in the `clearAll` function.

***Update:** Fixed in pull request #111.*

[N03] Undocumented side effect

The `withdrawWithTimelockFrom` function of the `PrizePool` contract puts the specified `amount` of tokens into a timelock. If the user already has time-locked tokens, their unlock deadline will be overwritten (all tokens will be in the same timelock), which is potentially an unexpected side effect. Consider documenting this possibility in the function comments.

***Update:** Fixed in pull request #112.*

[N04] Commented out code

The `initialize` function of the `MappedSinglyLinkedList` library contains a line of commented out code which only has debugging and testing purposes. As this line may confuse future developers and external contributors, consider removing it from the codebase.

***Update:** Fixed in pull request #103.*

[N05] Named return variables

There is an inconsistent use of named return variables across the entire code base. For instance, the functions in `CompoundPrizePool.sol` do not define a return variable in the function definition, but many of the functions in `PrizePool.sol` do.

Additionally, unused return variables are being declared in the `tokens`, `_currentTime`, `timelockBalanceAvailableAt`, `timelockBalanceOf`, and `accountedBalance`

Moreover, sometimes (for instance, in the `withdrawWithTimelockFrom` function of the `PrizePool` contract) the named return parameter is explicitly returned and other times (for instance, in the `withdrawInstantlyFrom` function) it is implicitly returned.

Consider removing all named return variables, explicitly declaring them as local variables, and adding the necessary return statements where appropriate. This should favor both explicitness and readability of the project.

Update: *Fixed in pull request #113.*

[N06] Typographical errors

- In `PrizePool.sol`:
- line 398: “addess” should be “address”
- line 422: “addess” should be “address”
- line 445: “addess” should be “address”
- line 584: “accounts” should be “account’s”
- In `ControlledToken.sol`:
- line 15: “Toen” should be “Token”

Update: *Fixed in pull request #114.*

Conclusions

1 critical and 2 high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

The PoolTogether team asked us to review the changes to the v3 contracts. We looked at the code and now publish our results.

Scope

The scope of this audit covers:

- A full review of the `CompoundPrizePoolBuilder`, `yVaultPrizePool` and `yVaultPrizePoolBuilder` contracts, and all the interfaces that are implemented by them in commit `5a9381340a7cef139c77e44c5ad5c2a5d6e12e36`.

The rest of the system was not included in the scope. All external code and contract dependencies were assumed to work as documented.

Update: *The Pooltogether team fixed in individual pull requests the issues we reported. We refer to these updates in their corresponding issues. Our analysis of the mitigations assumes the pull requests will be merged, but disregards any other potential changes to the code base.*

Overview of the changes

Overall, we are happy with the security posture of the team and the health of the codebase. As with the first phase of the v3 audit, we appreciate the small, encapsulated, and well-documented functions. We are also happy that the developers kept further generalizing the contract's design, which will make future upgrades easier.

This new phase introduces a new specialization of the `PrizePool` core component, the `yVaultPrizePool`, to be used with yEarn's yVaults instead of with Compound as the yield service. It also includes several refactors and additions to the codebase, such as a generic interface for yield services so that new services can be added in the future following the same specification as the existing ones, and builder contracts to easily create new prize pools for both Compound and yEarn yield services, together with new prize strategies.

Ecosystem dependencies

As mentioned in the first phase of the PoolTogetherv3 audit, understanding the external dependencies and assumptions has become a crucial component of the system's security.

Given that the audited changes introduce a new yield service to interact with, it must be noted that if these services are low on liquidity or behave unexpectedly, users may not be able to recover their deposits. The system should only be used if the yield service is trusted.

None.

High

None.

Medium

[M01] Capturing the award balance may fail

The `captureAwardBalance` function of the `PrizePool` contract relies on the underlying asset balance to calculate any available interest as award balance. Given that the way of calculating this balance may differ on each yield source, there is a possiblity that the calculated `totalInterest` may be greater than the `_currentAwardBalance` accumulated. For instance, the `balanceOfUnderlying` function in the Compound's `CToken` contract truncates the returned result, which could lead into an underflow when calculating the unnacounted prize balance in extreme cases, and therefore stall the function for future calls.

Even though this might not occur in the system as it is, this may change in future versions of the protocol when introducing new yield sources.

Consider checking that the `totalInterest` value calculated using the underlying asset's balance is greater than the accumulated `_currentAwardBalance`.

Update: *Fixed in pull request #205. The* `captureAwardBalance` *function now checks that the* `totalInterest` *value is greater than the* `_currentAwardBalance` *value to avoid the subtraction underflow.*

Low

[L01] Missing docstrings

All the functions in the `CompoundPrizePoolBuilder` and the `yVaultPrizePoolBuilder` contracts lack documentation. This hinders reviewers' can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Fixed in pull request #216.*

[L02] Tests not passing

The testing suite finishes with 4 failing tests. These tests are:

```
1) Tickets Feature: should not be possible to buy or transfer
tickets during award
2) SingleRandomWinner: removeExternalErc20Award()
3) SingleRandomWinner: addExternalErc721Award()
4) SingleRandomWinner: removeExternalErc721Award()
```

As the test suite was left outside of the audit's scope, please consider thoroughly reviewing the test suite to make sure all tests run successfully. Furthermore, it is advisable to only merge code that neither breaks the existing tests nor decreases coverage.

Update: *Fixed in pull request #205.*

Notes

[N01] `yVaultInterface` type mismatch

The `yVault` contract from yearn defines a `token` public variable, which will generate a public `token()` function, that will return the `IERC20` type instead of the `address` type defined on line 6 of `yVaultInterface.sol`.

Even though his does not pose a security risk, consider changing its type to `IERC20` to match the `yVault` contract specification, and to preserve consistency with other parts of the codebase.

The codebase implements the naming convention of prefixing `internal` values with an underscore. However, some functions and variables disregard this convention.

In the `PrizePool` contract:

- Consider renaming `calculateAccruedCredit` to `_calculateAccruedCredit`
- Consider renaming `tokenCreditPlans` to `_tokenCreditPlans`
- Consider renaming `tokenCreditBalances` to `_tokenCreditBalances`

In the `CompoundPrizePool` contract:

- Consider renaming `supplyRatePerBlock` to `_supplyRatePerBlock`

Additionally, in the `PrizePool` contract, consider renaming the `Initialized` event to `PrizePoolInitialized` or `CompoundPrizePoolCreated` as in the `CompoundPrizePoolBuilder` contract.

Update: *Fixed in pull request #207.*

[N03] Typographical errors

There are some typographical errors within the codebase:

- In line 946 of the `PrizePool` contract, “he pool” should be “the pool”
- In line 130 of the `yVaultPricePool` contract, “premptively” should be “preemptively”

Consider correcting these.

Update: *Fixed in pull request #208.*

[N04] Declare uint as uint256

To favor explicitness and consistency, consider declaring all instances of `uint` in the in `yVaultInterface` interface as `uint256`.

Update: *Fixed in pull request #208.*

practices and reduce the potential attack surface.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits