POWERINDEX SMART CONTRACTS AUDIT

(SCOPE 1)

November 25, 2020

PowerPool

MixBytes()

TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

General Provisions

PowerPool.Finance is a meta-governance protocol aimed at accumulating voting power in decentralized protocols and using it based on decisions of PowerPool (CVP) token holders.

The flagship product is PowerIndex - smart DeFi AMM pool for governance tokens.

PowerIndex consists of 8 governance tokens based on Balancer AMM. Project's team designed it to use pooled tokens for meta-governance (voting using pooled governance tokens) and fund management purposes (Vault-like strategies). It is entirely community-curated: CVP holders can add, remove GTs, change their weights, fees, and values of other index variables via governance proposals. The index has several "community" fees charged for the community needs and transferred to the treasury contract named "Permanent Voting Power".

Scope of the Audit

The scope of the audit includes the following smart contracts at:
PowerIndexPool.sol,
BPool.sol,
BToken.sol,
BMath.sol,
BNum.sol,
BConst.sol,
PoolRestrictions.sol

The audited commit identifier is: `37454e41f02b6c391b54357370689f73a48defbc`

02 SECURITY ASSESSMENT PRINCIPLES

Classification of Issues

- CRITICAL: Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

- MAJOR: Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

- WARNINGS: Bugs that can break the intended contract logic or expose it to DoS attacks.

- COMMENTS: Other issues and recommendations reported to/ acknowledged by the team.

Security Assessment Methodology

Two auditors independently verified the code.

Stages of the audit were as follows:

- "Blind" manual check of the code and its model
- "Guided" manual code review
- Checking the code compliance with customer requirements
- Discussion of independent audit results
- Report preparation

03 DETECTED ISSUES

CRITICAL

Not found

MAJOR

1. Event is logged with invalid arguments.

BToken.sol#L121
For the `Approval` event, the first argument is the sender's address and the second argument is the recipient's address.
The code needs to be changed from:
```
emit Approval (msg.sender, dst, _allowance [src] [msg.sender]);
```
to:
```
emit Approval (src, msg.sender, _allowance [src] [msg.sender]);
```

There are a lot of cases when wrong event badly affects the client's side sotfware (up to crash), so we recommend fixing it as soon as possible.

Status:

Fixed at PR-69
Client's commentary:

> This Issue was in the Balancer source code in BToken.sol, which was not modified by the PowerPool team for security reasons.

WARNINGS

1. Same origin transaction enables potential flash loan exploit.

This warning is about **BPool.sol** possibly enabling flash loan with allowing two and more transactions from the same `msg.sender` being processed at once in here:
BPool.sol#L463,
BPool.sol#L498,
BPool.sol#L535,
BPool.sol#L609,
BPool.sol#L684,
BPool.sol#L728,
BPool.sol#L773,
BPool.sol#L817,

For now the protection based on `tx.origin` is already implemented, however we recommend to add the check based on the same `msg.sender` .

Status:

Fixed at **PR-69**

2. Exact block timestamps usage.

This warning is about exact block timestamps being used to determine particular weight in here:
PowerIndexPool.sol#L192,
PowerIndexPool.sol#L189,
PowerIndexPool.sol#L76.

Dependency on particular block exact timestamps can eventually break the business logic because the timestamps not only may vary within the single replication time interval, but they can also be maliciously manipulated. As the result, the output of `_getDenormWeight` getter will vary every time it gets called.

It is recommended to refactor the business logic to be reliant not on a particular block timestamp, but on the replication interval time range. Usually it is about 15 seconds.
The more the better.

Status:

Acknowledged
Client's commentary:

> The Yellow Paper (Ethereum's reference specification) does not specify a constraint on the time period during which a block can drift, but it does specify that each timestamp should be bigger than the timestamp of its parent. Popular Ethereum protocol implementations Geth and Parity both reject blocks with the timestamp of more than 15 seconds in future. Therefore, a good rule of thumb in evaluating timestamp usage is: If the scale of your time-dependent event can vary by 15 seconds and maintain integrity, it is safe to use a `block.timestamp`.
>
> In our case, the most basic requirement is a smooth change in weights over time, which should take at least 5-7 days, depending on the current liquidity. Each next weight value must be guaranteed more/less than the previous one. Manipulation within 15 seconds over a time interval of 7 days (604800 seconds) cannot significantly change the weight and give any arbitration advantage to an attacker. On the other hand, if we use `block.number` as a base for weight calculation, Block time's increase due to the "Difficulty Bomb" launch can lead to a non-linear change in the weights.

3. Non-safe math usage.

This warning is about non-overflow resistant math being used to calculate the weight delta in `_getWeightPerSecond` in here: **PowerIndexPool.sol#L212**.

The absence of `bdiv` usage can lead to the contract failure with division to zero in case input `targetTimestamp` and `fromTimestamp` are both equal to zero.

It is recommended to introduce usage of `bdiv` for this particular function along with reimplementing **PowerIndexPool.sol#L212** particular place to eliminate the issue.

Status:

Fixed at PR-69
Client's commentary:

> `setDynamicWeight()` method checks that `targetTimestamp` and `fromTimestamp` can't be equal to zero: **PowerIndexPool.sol#L76-77**
> Nevertheless, this recommendation is implemented

4. Possible loss of tokens when they are sent to a zero address.

BToken.sol#L43
Add address verification needs to be done:

```
require(src != address(0), 'ERC20: transfer from the zero address');
require(dst != address(0), 'ERC20: transfer to the zero address');
```

BToken.sol#L89
Add address verification needs to be done:
`require(dst != address(0), 'ERC20: transfer to the zero address');`

BToken.sol#L95
Add address verification needs to be done:
`require(dst != address(0), 'ERC20: transfer to the zero address');`

BToken.sol#L101
Add address verification needs to be done:
`require(dst != address(0), 'ERC20: transfer to the zero address');`

Status:

Fixed at PR-69
Client's commentary:

> This Issue was in the Balancer source code in BToken.sol, which was not modified by the PowerPool team for security reasons.

5. The number of loop iterations should be limited.

The number of loop iterations is equal to the number of array elements. But the number of elements of this array is not limited from the top and theoretically can be very large. Since each iteration consumes gas, a situation may come to the point when there is not enough gas and the function will be blocked.

PoolRestrictions.sol#L69
For the array `_senders` and the function `setVotingAllowedForSenders`.

PoolRestrictions.sol#L77
For array `_addresses` and function `setWithoutFee`.

PoolRestrictions.sol#L109
For array `_poolsList` and function `_setTotalRestrictions`.

PoolRestrictions.sol#L119
For array `_signatures` and function `_setVotingSignatures`.

PoolRestrictions.sol#L134
For array `_signatures` and function `_setVotingSignaturesForAddress`.

We recommend checking the number of its iterations before executing the loops.

Status:

Fixed at PR-69
Client's commentary:

> These are admin functions that write data to mappings. If "out of gas" occurs, the transaction reverts without any harm except for the unnecessarily spent gas. Which is equivalent to revert due to check for the maximum number of iterations. Nevertheless, we implemented this recommendation.

COMMENTS

1.Uncontrolled access possible

BPool.sol#L960
If the variable `_wrapperMode` has the value `false`, then the access modifier function `_onlyWrapperOrNotWrapperMode` will allow any user to get access to the following functions:
`joinPool`, `exitPool`, `swapExactAmountIn`, `swapExactAmountOut`, `joinswapExternAmountIn`, `joinswapPoolAmountOut`, `exitswapPoolAmountIn`, `exitswapExternAmountOut`.
Also note that immediately after the contract is deployed, everyone has access to these functions.
It is imperative to disable this access in a separate transaction.

We recommend that you do not allow all users to access these features. Instead, you can use, for example, the following variable:

```
mapping (address => bool) private whitelist;
```

Only the owner of the contract will be able to manage such a variable and he will know who has access to this functionality.

Status:
No issue
Client's commentary:

> The primary purpose of the `_wrapperMode` variable is to prevent users from directly interacting with the PowerIndexPool.sol contract when needed. We disabled this feature by default.
>
> We may require this if we need to implement additional logic that does not fit in PowerIndex.sol due to its size.
>
> In this case, we will set the `_wrapperMode` to `true`, deploy the `PowerIndexPoolWrapper` contract, and set the `_wrapper` variable. `PowerIndexPoolWrapper` will contain additional logic, and through it, users will interact with the `PowerIndexPool`.

04 CONCLUSION AND RESULTS

Findings list

Level	Amount
CRITICAL	-
MAJOR	1
WARNING	5
COMMENT	1

Final commit identifier with fixes: `5609e5c832f329bfbd710d93f1b5148c84611e22`

About MixBytes

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

Disclaimer