PUBLIC REPORT

POA POSDAO Smart Contract Security Audit

peppersec

Foreword

Status of a vulnerability or security issue

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

No issue

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the label.

✓ Fixed

The fixed security issues get the label that informs those reading our public report that the flaws in question should no longer be worried about.

Addressed

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice tag right in front of it.

Acknowledged

If an issue is planned to be addressed in the future, it gets the tag, and a client clearly sees what is yet to be done.

Although the issues marked "Fixed" and "Acknowledged" are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

Severity levels

We also rank the magnitude of the risk a vulnerability or security issue pose. For this purpose, we use 4 "severity levels" namely:

1. Minor
2. Medium
3. Major
4. Critical

More details about the ranking system as well as the description of the severity levels can be found in **Appendix 1. Terminology**.

TABLE OF CONTENTS

04. STYLE

1. Logic separation Addressed
2. Signature parsing
3. Argument parsing
4. Reasons in require

CONCLUSION

APPENDIX 1. TERMINOLOGY

1. Severity

01. Introduction

1 Source code

Object	Location
DPOS	#fdaa685de851378b35741bdab986414dfd9042b1

2 Useful links

1. resourses (contracts, parity, tests, docs)
2. token distibution
3. WP (v1.4)
4. WP reviewes and QA

3 Security assessment methodology

The code of a smart contract has been automatically and manually scanned for known vulnerabilities and logic errors that may cause security threats. The conformity of requirements (e.g., White Paper) and practical implementation has been reviewed as well. More information on the used methodology can be found here.

4 Auditors

1. Alexey Pertsev
2. Roman Storm
3. Anton Bukov

02. Summary

Below, you can find a table with all the discovered bugs and security issues listed.

Security issue	Severity
Redundant modifier	Major
Reward mechanism	
Rounding errors	
Random validators algorithm problem	
collectRoundLength should have more restrictions	Medium
Possible weak random using _getRandomIndex	
Reward mechanism	
Members' hashes access	
Reward function optimization	
transferOwnership + claimOwnership	
Token stake and withdraw methods	Minor
Consider extracting pool structure	
require for upgradeTo	
Allowed senders storing	
Hardcoded value	

03. General issues

1 Redundant modifier

Severity: MAJOR

Redundant onlyValidatorSetContract modifier forbids using getCurrentSeed by any DApp in the network.

Note: Even though it would be great for any Dapp to be capable of obtaining random numbers from the network, there are some limitations the developer should take into account:

1. The new random number happens only in particular blocks (it depends on the network configuration) at the end of the collect round.
2. The revealing validator always knows the next random number before sending. So, Dapp should restrict any business logic action that depends on random during the reveal phase

Recommendations:

1. Consider removing the modifier.
2. Provide a smart contract example of using getCurrentSeed by a third-party Dapp.

Status:

✓ Fixed Link

2 collectRoundLength should have more restrictions

Severity: MEDIUM

According to the code, the `_collectRoundLength` variable should be even and more than zero.

Recomendations:

Consider adding a couple more restrictions to facilitate system hardening:

1. `_collectRoundLength % validators_count === 0` to eleminate validator cartels
2. `stakingEpoch % _collectRoundLength === 0` to ensure every validator can take part of random generation in the last block of an epoch.

Status:

✓ Fixed Initializer **fix**, related code **fix**.

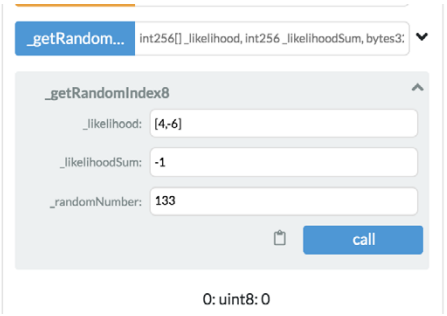
3 Possible weak random using _getRandomIndex

Severity: MEDIUM

The _getRandomIndex(int256[] memory _likelihood, int256 _likelihoodSum, uint256 _randomNumber) function is used to get a random index for validators array using _randomNumber.

If _likelihoodSum is a small negative value, then _getRandomIndex always returns 0 for some range of the _randomNumber values.

Take a look at the example with int8:




```
26  function _getRandomIndex8(int8[] memory _likelihood, int8 _likelihoodSum, uint8 _randomNumber)
27      public
28      pure
29      returns(uint8)
30  {
31      int8 r = int8(_randomNumber % uint8(_likelihoodSum)) + 1;
32      uint8 index = 0;
33      while (true) {
34          r -= _likelihood[index];
35          if (r <= 0) break;
36          index++;
37      }
38      return index;
39  }
40
41  function convert(int256 _likelihoodSum)
```

"r" is negative before for loop due to overflow during int to uint converting

On the other hand, according to the _setLikelihood method, it is not possible right now to have a negative 'likelihood' because stakeAmountTotalMinusOrderedWithdraw never returns a negative value.

Recommendations:

Please, have a look at the examples on the next page.

```
1   function _getURandomIndex1(uint256[] memory _likelihood, uint256 _likelihoodSum,
    uint256 _randomNumber)
2       public
3       pure
4       returns(uint256)
5   {
6       uint256 random = _randomNumber % _likelihoodSum;
7       uint256 sum = 0;
8       uint256 index = 0;
9       while (sum <= random) {
10          sum += _likelihood[index];
11          index++;
12      }
13      return index-1;
14  }
15
16  function _getURandomIndex2(uint256[] memory _likelihood, uint256 _likelihoodSum,
    uint256 _randomNumber)
17      public
18      pure
19      returns(uint256)
20  {
21      uint256 random = (_randomNumber % _likelihoodSum) + 1;
22      uint256 index = 0;
23      while (true) {
24          uint256 weight = _likelihood[index];
25          if(random > weight) {
26              random -= weight;
27          } else {
28              break;
29          }
30          index++;
31      }
32      return index;
33  }
```

Status

✓ Fixed Link

4 Reward mechanism

Severity: **MAJOR**

Paying block rewards in loops is not the safest thing to do. Also, we suspect that per-epoch reward payments, snapshots, and unlimited gas blocks were introduced to fix this issue.

Recommendations:

1. I would suggest reading about fees rewards in projects:
   - Bancor Network
   - Uniswap Exchange
   - Compound Finance (cETH, cDAI, etc.)
   - Callisto Cold Staking
2. Look at the example of **the StakingPool implementations.**

Status:

Acknowledged The team is working on a new implementation.

5 Members' hashes access

Severity: **MEDIUM**

The constants with the hashes of members' names are used unsafely in multiple smart contracts. In every method, we should not forget about a proxy pattern, which leads to unexpected mistakes.

Recommendations:

1. The constants with the hashes of members' names could be extracted to the parent contract and made `private` instead of `internal`.
2. Add `internal` and/or `public` getters and/or setters to introduce an API for an inherited contract.

Status:

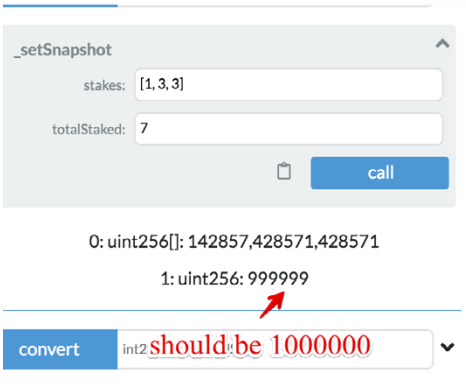
Addressed The team decided to migrate to a new upgradable pattern. So, the issue will be automatically fixed in the new version.

6 Rounding errors

Severity: **MAJOR**

A rounding error occurs in the `_setSnapshot` during reward percentages distribution. It can lead to unexpected results.

Here is a simplified example with the same `distribution` logic:




```
26      uint256 REWARD_PERCENT_MULTIPLIER = 1000000;
27      function _setSnapshot(uint256[] memory stakes, uint256 totalStaked)
28          public
29          view
30          returns(uint256[] memory, uint256)
31      {
32          uint256 rewardPercent = 0;
33          uint256 rewardPercentsSum = 0;
34          uint256[] memory rewardPercents = new uint256[](stakes.length);
35          for (uint256 i = 0; i < stakes.length; i++) {
36              rewardPercent = stakes[i];
37              rewardPercent = REWARD_PERCENT_MULTIPLIER * rewardPercent / totalStaked;
38              rewardPercents[i] = rewardPercent;
39              rewardPercentsSum += rewardPercent;
40          }
41          return (rewardPercents, rewardPercentsSum);
42      }
```

As you can see, rewardPercentSum is not equal to 100% as it is supposed to.

Recommendations:

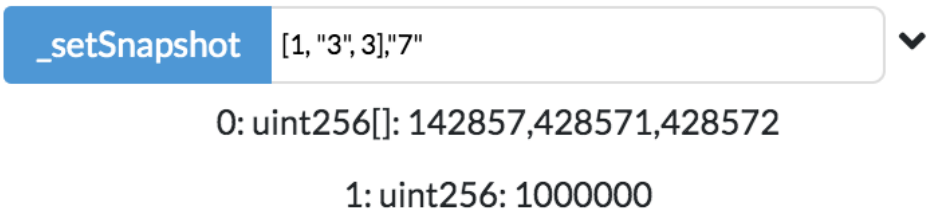
The example above could be fixed in the way specified on the next page.

```
uint256 REWARD_PERCENT_MULTIPLIER = 1000000;
function _setSnapshot(uint256[] memory stakes, uint256 totalStaked)
    public
    view
    returns(uint256[] memory, uint256)
{
    uint256 rewardPercent = 0;
    uint256 rewardPercentsSum = 0;
    uint256[] memory rewardPercents = new uint256[](stakes.length);
    for (uint256 i = 0; i < stakes.length - 1; i++) {
        rewardPercent = stakes[i];
        rewardPercent = REWARD_PERCENT_MULTIPLIER * rewardPercent / totalStaked;
        rewardPercents[i] = rewardPercent;
        rewardPercentsSum += rewardPercent;
    }

    uint256 roundedPercent = REWARD_PERCENT_MULTIPLIER - rewardPercentsSum;
    uint256 lastIndex = stakes.length - 1;
    rewardPercents[lastIndex] = roundedPercent;
    rewardPercentsSum += roundedPercent;

    // just to be sure
    rewardPercent = stakes[lastIndex];
    rewardPercent = REWARD_PERCENT_MULTIPLIER * rewardPercent / totalStaked;
    assert(rewardPercent <= roundedPercent);

    return (rewardPercents, rewardPercentsSum);
}
```



Status:

✓ Fixed Link

7 Token stake and withdraw methods

Severity: **MINOR**

The token implementation contains special methods for staking contract interaction. While the `stake()` method allows for performing `transferFrom` without `approve()` to the staking contract, the second method is just an equal to `transfer`.

```
function stake(address _staker, uint256 _amount) external onlyStakingContract {
    // Transfer `_amount` from `_staker` to `stakingContract`
    require(_amount <= balances[_staker]);
    balances[_staker] = balances[_staker].sub(_amount);
    balances[stakingContract] = balances[stakingContract].add(_amount);
    emit Transfer(_staker, stakingContract, _amount);
}

function withdraw(address _staker, uint256 _amount) external onlyStakingContract {
    // Transfer `_amount` from `stakingContract` to `_staker`
    require(_amount <= balances[stakingContract]);
    balances[stakingContract] = balances[stakingContract].sub(_amount);
    balances[_staker] = balances[_staker].add(_amount);
    emit Transfer(stakingContract, _staker, _amount);
}
```

Recommendations:

We would recommend using the ERC20 default scheme with `approve + transferFrom`. Also, both methods contain redundant requirements, which can be omitted since SafeMath is used.

Status:

Acknowledged Link to partial fix

8 Consider extracting pool structure

Severity: **MINOR**

Pools are stored in the map and have reverse lookup. Perhaps it is worth trying to extract this behavior into a separate struct and keeping it proxy-compatible? Here is an example of Set with reverse lookup.

Status:

Acknowledged

9 Reward function optimization

Severity: **MEDIUM**

This logic can be executed as a part of other calculations later.

Recommendations:

1. Consider reducing the number of for-loops in the reward function.

Status:

Acknowledged

10 Random validators algorithm problem

Severity: **MAJOR**

The current algorithm pseudocode:

```
for (uint j = 0; j < 19; j++) {
    uint value = nextRandom() % sumLikehood;
    for (uint i = 0; i < n - j; i++) {
        if (value < candidates[i].likehood) {
            validators[j] = i;
            sumLikehood -= candidates[i].likehood;
            candidates[i] = candidates[candidates.length - 1 - j];
            break;
        }
        value -= candidates[i].likehood;
    }
}
```

It provides the distrubution that does not depend on the order of candidates:

```
Experiment: 2 validators, 5 candidates, 1000000 simulations, sort ascending
Candidate 2: weight 106, probability of being selected 0.068895
Candidate 0: weight 242, probability of being selected 0.151489
Candidate 1: weight 311, probability of being selected 0.189437
Candidate 4: weight 406, probability of being selected 0.238352
Candidate 3: weight 739, probability of being selected 0.351826

Experiment: 2 validators, 5 candidates, 1000000 simulations, sort descending
Candidate 2: weight 106, probability of being selected 0.069112
Candidate 0: weight 242, probability of being selected 0.151018
Candidate 1: weight 311, probability of being selected 0.189736
Candidate 4: weight 406, probability of being selected 0.238694
Candidate 3: weight 739, probability of being selected 0.351440
```

But the probabilities are shifted from the original ones:

```
0.058758 // weight 106
0.134146 // weight 242
0.172395 // weight 311
0.225055 // weight 406
0.409645 // weight 739
```

You may notice that smaller pools lead to higher reward distribution. This will result in candidates creating multiple min-stake virtual pools instead of staking all their money on a single own pool.

We also tried different algorithms, including this **one**:

```python
import heapq
import math
import random

def WeightedSelectionWithoutReplacement(weights, m):
    elt = [(math.log(random.random()) / weights[i], i) for i in range(len(weights))]
    return [x[1] for x in heapq.nlargest(m, elt)]
```

But **the produced results** were pretty much the same.

So, we have come up with a new solution – to introduce weights of validators with the following algorithm:

```
std::vector<int> validators(m);
std::vector<int> weights(n);
int totalWeights = 0;

for (int j = 0; j < m; j++) {
    int value = rand() % sum;
    for (int i = 0; i < n; i++) {
        if (value < likehoods[i]) {
            if (weights[i] == 0) {
                validators[j] = i;
            } else {
                j--;
            }
            weights[i]++;
            totalWeights++;
            break;
         }

        value -= likehoods[i];
    }
}
```

It provides us with the following results:

```
Experiment: 2 validators, 5 candidates, 1000000 simulations, sort ascending
Candidate 2: weight 106, probability of being selected 0.058748
Candidate 0: weight 242, probability of being selected 0.134218
Candidate 1: weight 311, probability of being selected 0.172222
Candidate 4: weight 406, probability of being selected 0.225277
Candidate 3: weight 739, probability of being selected 0.409534

Experiment: 2 validators, 5 candidates, 1000000 simulations, sort descending
Candidate 2: weight 106, probability of being selected 0.058303
Candidate 0: weight 242, probability of being selected 0.134276
Candidate 1: weight 311, probability of being selected 0.172492
Candidate 4: weight 406, probability of being selected 0.225072
Candidate 3: weight 739, probability of being selected 0.409857
```

Recommendations:

1. Consider implementing the proposed algorithm
2. Just distribute reward between 19 validatros proportinally to new weights (1,2,3...).

Status:

Acknowledged

11 TransferAndCall does not restrict transfers to stakingContract

Severity: **MAJOR**

`transferAndCall` does not restrict transfers to the staking contract (`transfer` and `transferFrom` do). This token mock is actually ported from **poa-bridge-contracts**. So, the issue should be addressed first among others to be addressed.

Recommendations:

1. Consider restricting all of the ways to transfer tokens to `stakingContract`.

- ☑ transfer
- ☑ transferFrom
- ☐ transferAndCall

Optional:

- ☐ mint
- ☐ claimTokens

Status:

Acknowledged

12 transferOwnership + claimOwnership

Severity: **MEDIUM**

Perhaps, it will make sense to implement the `transferOwnership` + `claimOwnership` scheme instead of just `transferOwnership` without proper knowledge, just like a new owner could do.

Status:

Acknowledged Will be implemented

13 require for upgradeTo

Severity: **MINOR**

Is there any purpose of using “`return false` in case of fail” instead of `require(someCheck(), “useful message”)` for `upgradeTo` method?

Status:

Addressed Will be updated with an upgradable pattern

14 Allowed senders storing

Severity: **MINOR**

TXPermission. Why are allowed senders stored as array? It seems, mapping would be more efficient.

Status:

Addressed Will be updated in a new contract version

15 Hardcoded value

Severity: **MINOR**

Is **this value** hardcoded for a reason?

Status:

Addressed Will be replaced with MAX_VALIDATORS

04. Style

1 Logic separation

To avoid too deep stack situations and increase readability, we highly recommend keeping function less than 50 lines of code.

Recommendations:

Consider refactoring reward, _distributeRewards, and other functions longer than 50 lines of code to set off internal functions with readable names.

Status:

Addressed Will be entirely removed.

2 Signature parsing

The code

```
for (i = 0; _data.length >= 4 && i < 4; i++) {
    signature |= bytes4(_data[i]) >> i*8;
}
```

could be replaced with the following:

```
assembly {
    signature := shl(224, mload(add(_data, 4)))
}
```

Status:

Acknowledged

3 Argument parsing

The code

```
abiParams = new bytes(_data.length - 4 > 32 ? 32 : _data.length - 4);

for (i = 0; i < abiParams.length; i++) {
    abiParams[i] = _data[i + 4];
}

if (signature == bytes4(keccak256("commitHash(bytes32,bytes)"))) {
    (bytes32 secretHash) = abi.decode(abiParams, (bytes32));
}
```

could be replaced with this:

```
bytes32 secretHash;
if (signature == bytes4(keccak256("commitHash(bytes32,bytes)"))) {
    uint256 secretHashOffset = 36;
    assembly {
        secretHash := mload(add(_data, secretHashOffset))
    }
}
```

Status:

Acknowledged

4 Reasons in require

That would be nice to have a readable reason of reverted transaction. Even though none of explorers shows a reason, it could be retrieved using `eth_call`.

Status:

The team does not use the reasons in require to keep the contract's size as small as possible.

Conclusion

Despite the fact the auditors have discovered 4 MAJOR severity vulnerabilities that could be fixed, the audit and stress testing indicate that the current architecture is not scalable and prone to attacks. Together, both teams are elaborating on a new architecture where stakers will accumulate and later “pull” their stakes and rewards instead of the “push” strategy as it is implemented now.
Since some fixes require too many changes in code, the Peppersec team recommends holding a new audit before the release.

Appendix 1. Terminology

1 Severity

Severity is the category that described the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood		

MINOR

Minor issues are generally subjective in their nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

MEDIUM

Medium issues are generally objective in their nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is an apparent reason not to.

MAJOR

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

CRITICAL

Critical issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems and ultimately a full failure in the operations of the contract.

About Us

Worried about the security of your project? You're on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you've just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We're ready to battle for the security of your project.

LET'S KEEP IN TOUCH


peppersec.com


hello@peppersec.com


Github