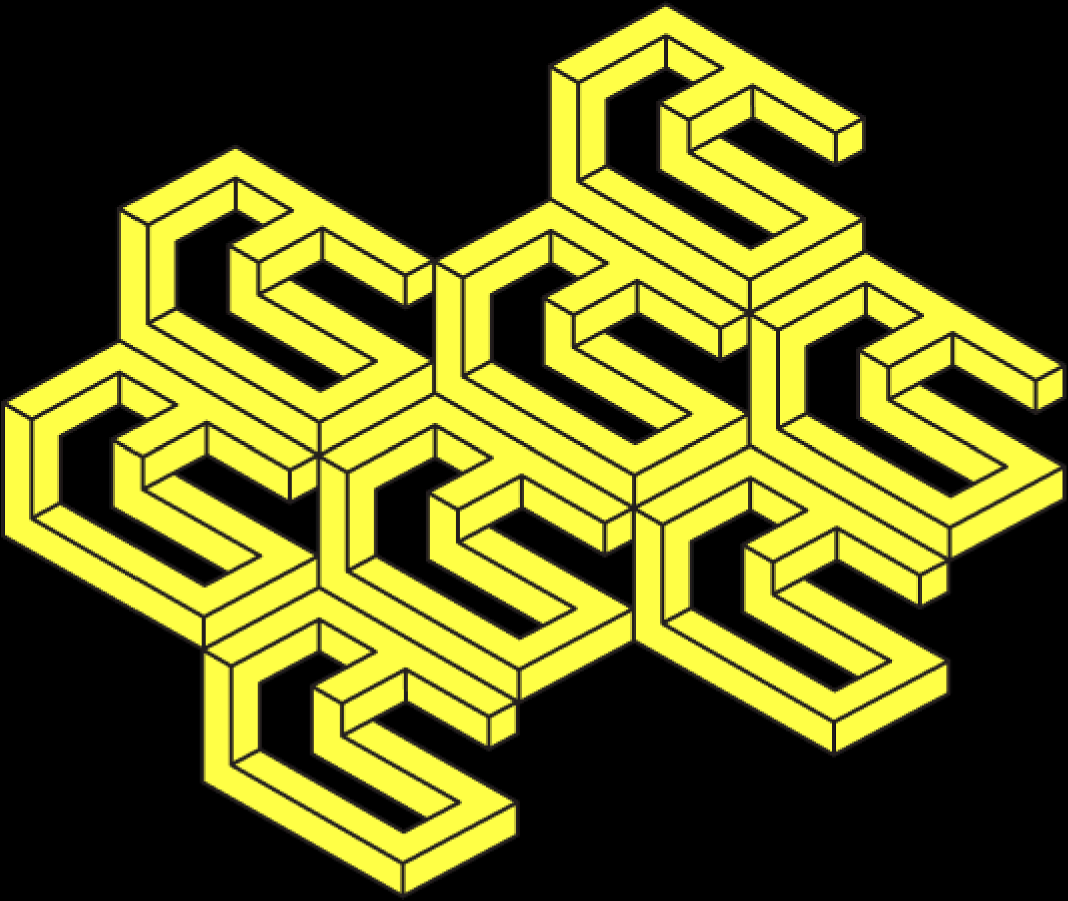

Rocket Pool DAO Smart Contracts Update Review | April 2024

by ChainSafe Systems | April 2024

Table of contents

- 1. Introduction
  - Defining Severity
    - Referencing updated code
    - Disclaimer
- 2. Executive Summary
- 3. Critical Bugs and Vulnerabilities
- 4. Line-by-line review
  - contracts/contract/RocketBase.sol
  - contracts/contract/dao/protocol/RocketDAOProtocol.sol
  - contracts/contract/dao/protocol/RocketDAOProtocolProposal.sol
  - contracts/contract/dao/protocol/RocketDAOProtocolProposals.sol
  - contracts/contract/dao/protocol/RocketDAOProtocolVerifier.sol
  - contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsNode.sol
  - contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsSecurity.sol
  - contracts/contract/dao/security/RocketDAOSecurityProposals.sol
  - contracts/contract/network/RocketNetworkPrices.sol
  - contracts/contract/network/RocketNetworkSnapshots.sol
  - contracts/contract/network/RocketNetworkVoting.sol
  - contracts/contract/node/RocketNodeStaking.sol
  - contracts/contract/upgrade/RocketUpgradeOneDotThree.sol
  - contracts/contract/rewards/RocketClaimDAO.sol

1. Introduction

Date	Auditor(s)
April 2024	Oleksii Matiiasevych, Anderson Lee

Rocket Pool Pty Ltd requested **ChainSafe Systems** to perform a review of the Rocket Pool DAO smart contracts update. The contracts can be identified by the following git diff:

`6a9dbfd85772900bb192aabeb0c9b8d9f6e019d1 original`
`60684a7f0366a4233164a4d264b70991cc3cd86f update`

There are 72 contracts, interfaces and libraries in scope.

After the initial review, Rocket Pool team applied a number of updates which can be identified by the following git commit hash:

`84ac19872dda7ca9c39c4f7349159d0e984130b9`

Additional verification was performed after that.

Defining Severity

Each finding is assigned a severity level.

Note	Notes are informational in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
Optimization	Optimizations are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Minor	Minor issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Major	Major 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.
Critical	Critical issues are directly exploitable security vulnerabilities that need to be fixed.

Referencing updated code

Resolved	The finding has been acknowledged and the team has since updated the code.
Rejected	The team dismissed this finding and no changes will be made.

Disclaimer

The review makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts for any specific purpose, or their bug free status.

2. Executive Summary

All the initially identified minor and above severity issues were fixed and are not present in the final version of the contracts.

There are **no** known compiler bugs for the specified compiler version (0.8.18), that might affect the contracts’ logic.

There were **1 critical**, **2 major**, **1 minor**, 32 informational/optimization issues identified in the initial version of the contracts. The Rocket Pool team provided a comprehensive documentation complemented with illustrations on how the voting protocol is supposed to work, which allowed us to fully comprehend and validate the logic behind it.

We are looking forward to future engagements with the Rocket Pool.

3. Critical Bugs and Vulnerabilities

Single critical issue RocketDAOProtocolVerifier.getPollardRootIndex() was identified in the contracts which could allow a malicious actor to create invalid yet unchallengeable proposals technically taking control of the DAO.

4. Line-by-line review

contracts/contract/RocketBase.sol

L16 Optimization Rejected

```
uint8 public version;
```

The `version` state variable could be made immutable.

L19 Optimization Rejected

```
RocketStorageInterface rocketStorage =
RocketStorageInterface(address(0));
```

The `rocketStorage` state variable could be made immutable.

contracts/contract/dao/protocol/RocketDAOProtocol.sol

L55 Optimization Rejected

```
function bootstrapSettingMulti(...) ...
onlyLatestContract("rocketDAOProtocol", address(this)) {
```

The `bootstrapSettingMulti()` function has an `onlyLatestContract` modifier that checks itself, which is excessive assuming that the inside call to `proposalSettingMulti()` has the same modifier applied. Same applies to other infrastructure functions in the repository.

contracts/contract/dao/protocol/RocketDAOProtocolProposal.sol

L5 Note Resolved

The `RocketDAOProtocolInterface` import could be removed.

L6 Note Resolved

The `RocketDAOProtocolProposalsInterface` import could be removed.

L7 Note Resolved

The `RocketDAOProtocolSettingsInterface` import could be removed.

L8 Note Resolved

The `RocketDAOProtocolSettingsRewardsInterface` import could be removed.

L9 Note Resolved

The `RocketClaimDAOInterface` import could be removed.

L10 Note Resolved

The `RocketDAOProposalInterface` import could be removed.

L11 Note Resolved

The `RocketNodeManagerInterface` import could be removed.

L12 Note Resolved

The `SettingType` import could be removed.

L50-52 Optimization Resolved

```
for (uint256 i = 0; i < _treeNodes.length; i++) {
    totalVotingPower += _treeNodes[i].sum;
}
```

The `_treeNodes.length` could be cached with a local variable.

L354 Note Resolved

```
require(_blockNumber <= block.number, "Block must be in the past");
```

The `_propose()` function allows creation of proposals at the current block which is prone to be frontrun invalidating the voting power Merkle tree. For instance a malicious actor could change their voting power.

contracts/contract/dao/protocol/RocketDAOProtocolProposals.sol

L50 Note Resolved

```
function proposalSettingMulti(...) ... onlyExecutingContracts() {
```

The `onlyExecutingContracts` modifier has parentheses at the end even though other modifiers don't.

contracts/contract/dao/protocol/RocketDAOProtocolVerifier.sol

L186 Note Rejected

```
require(depth < maxDepth * 2, "Invalid index depth");
```

The `createChallenge()` function validates the index `depth < maxDepth * 2` twice. First in the beginning of the function and second in the `getPollardRootIndex()` function.

L289 Note Resolved

The `claimBondChallenger()` function should not expect a situation where some challenges are still unresponded, and the proposal is not defeated at the same time.

L393-396 Major Resolved

```
uint256 state = getUint(challengeKey);

// Make sure this index was actually challenged
require(state != 0, "Challenge does not exist");
```

The `submitRoot()` function allows a proposer to resubmit `root` multiple times, resetting the challenge state from `Paid` to `Responded`, then unlocking their stake multiple times.

L466 Optimization Rejected

```
uint256 delegateIndex = (_offset + i) / nodeCount;
```

The `verifyLeaves()` function excessively calculates `delegateIndex` for every leaf, while the resulting index is the same for all of them.

L474 Optimization Rejected

```
if (actualDelegate == rocketNodeManager.getNodeAt(delegateIndex)) {
```

The `verifyLeaves()` function excessively calls `rocketNodeManager.getNodeAt(delegateIndex)` for every leaf, while the resulting address is the same for all of them.

L475 Major Resolved

```
actual = rocketNetworkVoting.getVotingPower(nodeAddress, blockNumber32);
```

The `verifyLeaves()` function could produce different results during proposal lifetime based on the `node.per.minipool.stake.maximum` setting value. This would invalidate any pending proposal.

L633-649 Critical Resolved

```
if (indexDepth < maxDepth) {
    // Index is leaf of phase 1 tree
    uint256 remainder = indexDepth % depthPerRound;
    require(remainder == 0, "Invalid index");
    return _index / (2 ** depthPerRound);
} else if (indexDepth == maxDepth) {
    // Index is a network tree leaf
    uint256 remainder = indexDepth % depthPerRound;
    return _index / (2 ** remainder); // <- Critical Issue
} else if (indexDepth < maxDepth * 2) {
    // Index is phase 2 pollard
    uint256 subIndexDepth = indexDepth - maxDepth;
    uint256 remainder = subIndexDepth % depthPerRound;
    require(remainder == 0, "Invalid index");
    return _index / (2 ** depthPerRound);
}
revert("Invalid index");
```

The `getPollardRootIndex()` produces invalid results at the bottom of the network tree when `maxDepth % depthPerRound == 0`. This would result in a proposal being unchallengeable. The following formula is incorrect `_index / (2 ** remainder)` and should be changed to something like `_index / (2 ** (remainder == 0 ? depthPerRound : remainder))`.

contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsNode.sol

L10 Note Resolved

The `RocketDAOProtocolSettingsNode` contract doesn't have set settings validation conditions.

contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsSecurity.sol

L16-35 Note Resolved

```
setSettingUint("members.leave.time", 4 weeks);
...
} else if(settingKey ==
keccak256(abi.encodePacked("members.leave.time"))) {
    // < 14 days (RPIP-33)
    require(_value < 14 days, "Value must be < 14 days");
```

The `constructor()` sets the `members.leave.time` at 4 weeks, while the `setSettingUint()` function has a less than 2 weeks condition.

contracts/contract/dao/security/RocketDAOSecurityProposals.sol

L174 Optimization Rejected

```
setBool(keccak256(abi.encodePacked(daoNameSpace, "member",
_memberAddress)), false);
```

The `_memberInit()` function excessively sets `member` to `false`, while it should already be `false`.

L177 Optimization Rejected

```
setUint(keccak256(abi.encodePacked(daoNameSpace, "member.joined.time",
_memberAddress)), 0);
```

The `_memberInit()` function excessively sets `member.joined.time` to `0`, while it should already be `0`.

contracts/contract/network/RocketNetworkPrices.sol

L14 Optimization Resolved

```
bytes32 priceKey;
```

The `priceKey` state variable could be made immutable.

L15 Optimization Resolved

```
bytes32 blockKey;
```

The `blockKey` state variable could be made immutable.

contracts/contract/network/RocketNetworkSnapshots.sol

L115 Optimization Resolved

```
result._value = uint224(uint256(raw) & (2 ** 224 - 1));
```

The `_load()` function applies a 224 bit mask to the raw value, then casting it into a `uint224`. Masking is excessive, just casting is enough as the compiler performs masking by itself.

L128 Optimization Resolved

```
return uint224(uint256(raw) & (2 ** 224 - 1));
```

The `_valueAt()` function applies a 224 bit mask to the raw value, then casting it into a `uint224`. Masking is excessive, just casting is enough.

contracts/contract/network/RocketNetworkVoting.sol

L109 Note Resolved

```
uint256 rplStake = uint256(rocketNetworkSnapshots.lookupRecent(key,
uint32(_block), 5));
```

The `getVotingPower()` function excessively casts `_block` to `uint32`.

contracts/contract/node/RocketNodeStaking.sol

L88-91 Optimization Resolved

```
uint256 value = getUint(key);
setUint(key, value + _amount); // Optimization Issue
RocketNetworkSnapshotsInterface rocketNetworkSnapshots =
RocketNetworkSnapshotsInterface(getContractAddress("rocketNetworkSnapshot
s"));
rocketNetworkSnapshots.push(key, uint32(block.number), uint224(value +
_amount));
```

The `increaseNodeRPLStake()` function keeps updating the deprecated storage slot along with a snapshot even if the node stake value already migrated to snapshots.

L98-101 Optimization Resolved

```
uint256 value = getUint(key);
setUint(key, value - _amount); // Optimization Issue
RocketNetworkSnapshotsInterface rocketNetworkSnapshots =
RocketNetworkSnapshotsInterface(getContractAddress("rocketNetworkSnapshot
s"));
rocketNetworkSnapshots.push(key, uint32(block.number), uint224(value -
_amount));
```

The `decreaseNodeRPLStake()` function keeps updating the deprecated storage slot along with a snapshot even if the node stake value already migrated to snapshots.

L413-417 Minor Resolved

```
uint256 rplStake = getNodeRPLStake(_nodeAddress);
uint256 lockedStake = getNodeRPLLocked(_nodeAddress);
require(rplStake >= _amount, "..."); // Minor Issue
// Check withdrawal would not under collateralise node
require(rplStake - _amount - lockedStake >=
getNodeMaximumRPLStake(_nodeAddress), "...");
```

The `withdrawRPL()` function does not account for `lockedStake` when validating the amount. The requirement should be: `rplStake >= _amount + lockedStake`.

contracts/contract/upgrade/RocketUpgradeOneDotThree.sol

L279 Note Resolved

The `_deleteContract()` function is not used.

L288 Note Resolved

The `_upgradeABI()` function is not used.

contracts/contract/rewards/RocketClaimDAO.sol

L151-153 Optimization Resolved

```
setSettingUint("members.leave.time", 4 weeks);
...
for (uint256 i = 0; i < _contractNames.length; i++) {
    payOutContract(_contractNames[i]);
}
```

The `_contractNames.length` could be cached with a local variable.