

Rocket Pool Atlas (v1.2)

1 Executive Summary

1.1 Mitigations: 17 Mar 2023

2 Scope

2.1 Objectives

3 Document Change Log

4 System Overview

5 Findings

5.1 RocketNodeDistributorDelegate - Reentrancy in `distribute()` allows node owner to drain distributor funds **Critical**
✓ Fixed

5.2 RocketMinipoolDelegateOld - Node operator may reenter `finalise()` to manipulate accounting **Major** ✓ Fixed

5.3 RocketMinipoolDelegate - Sandwiching of Minipool calls can have unintended side effects **Major**

5.4 RocketDAONodeTrustedActions - No way to access ETH provided by non-member votes **Major**
Acknowledged

5.5 Multiple checks-effects violations **Major**

5.6 Minipool state machine design and pseudo-states **Medium**
Acknowledged

5.7 RocketMinipoolDelegate - Redundant `refund()` call on forced finalization **Medium**
✓ Fixed

5.8 Sparse documentation and accounting complexity **Medium**
Acknowledged

5.9 RocketNodeDistributor - Missing `extcodesize` check in dynamic proxy **Medium**
Won't Fix

5.10 Kicked oDAO members' votes taken into account **Medium**
Acknowledged

5.11 RocketDAOProtocolSettingsRewards - settings key collision **Medium**
Acknowledged

5.12 RocketDAOProtocolSettingsRewards - missing setting delimiters **Medium** Acknowledged

5.13 Use of `address` instead of specific contract types **Minor**
Acknowledged

5.14 Redundant double casts **Minor** Acknowledged

5.15 RocketMinipoolDelegate - Missing event in `prepareVacancy` **Minor** ✓ Fixed

5.16 Compiler error due to missing `RocketMinipoolBaseInterface` **Minor** ✓ Fixed

5.17 Unused Imports **Minor**
Partially Addressed

5.18 RocketMinipool - Inconsistent access control modifier

Date	January 2023
Auditors	Dominik Muhs, Martin Ortner

1 Executive Summary

This report presents the results of our engagement with **Rocket Pool** to review their upcoming Rocket Pool Atlas release (v1.2).

The review was conducted over three weeks, from **January 16, 2023** to **February 03, 2023**. A total of 2x15 person-days were spent.

A critical reentrancy issue in the node distributor has been found, allowing a node owner to drain funds from the respective distributor. Furthermore, several major severity issues have been found regarding the updated Minipool delegate contract and a node operator DAO contract.

1.1 Mitigations: 17 Mar 2023

Latest commit with changes: [rocket-pool/rocketpool@77d7cca](#)

Compared to audit commit: [rocket-pool/rocketpool@7771afa...77d7cca](#)

The client provided code changes and remediation information for the findings outlined in this report. Details and can be found in the Remediation Notes for the respective findings in [Section: Findings](#).

- We removed one recommendation noting that zero initialization is unnecessary agreeing with and following the clients remark that "[...]Being explicit even if redundant is not a problem."
- We downgraded the finding "Missing `extcodesize` check in dynamic proxy" from **Major** -> **Medium** in agreement with the client as funds are not at immediate risk and they can recover from this problem. Nevertheless, this finding should be addressed as per the recommendation.
- We downgraded the finding "Duplicate check to avoid revert" from **Minor** -> **Informational** given that it more a technical debt than a concrete security issue and the client provided a the following out-of-band fix: [rocket-pool/rocketpool@3ab7af1](#)

2 Scope

Our review focused on the following repositories:

- [rocketpool@7771afa...fde3f8c](#)

Furthermore, the following information was provided:

- [GitHub/Gist describing changes](#) (accessed January 2023)

2.1 Objectives

Together with the client, we identified the following priorities for our review:

- Ensure that the system is implemented consistently with the intended functionality and without unintended edge cases.
- Identify known vulnerabilities particular to smart contract systems, as outlined in our [Smart Contract Best Practices](#), and the [Smart Contract Weakness Classification Registry](#).
- Review key risks in regard to recent changes flagged by the development team:
 - The minipool delegate has changed, and the possible state transitions are more complex.
 - The deposit credit system used for the lower ETH bonded minipools and solo migration is new
 - The minipool distribution (reward distribution) logic has changed considerably to incorporate partial withdrawals (skimming), optimizing withdrawals, and solo migration. With the Shanghai hard fork, this will be the first time these features will be exercised in production.

3 Document Change Log

Version	Date	Description
1.0	2023-02-06	Initial report
1.1	2023-03-17	Updated Report: Mitigations
1.2	2023-03-24	Updated Report: Client provided a fix for 5.2

4 System Overview

declaration onlyMiniPoolOwner

Minor Acknowledged


5.19 RocketDAO*Settings -
settingNameSpace should be

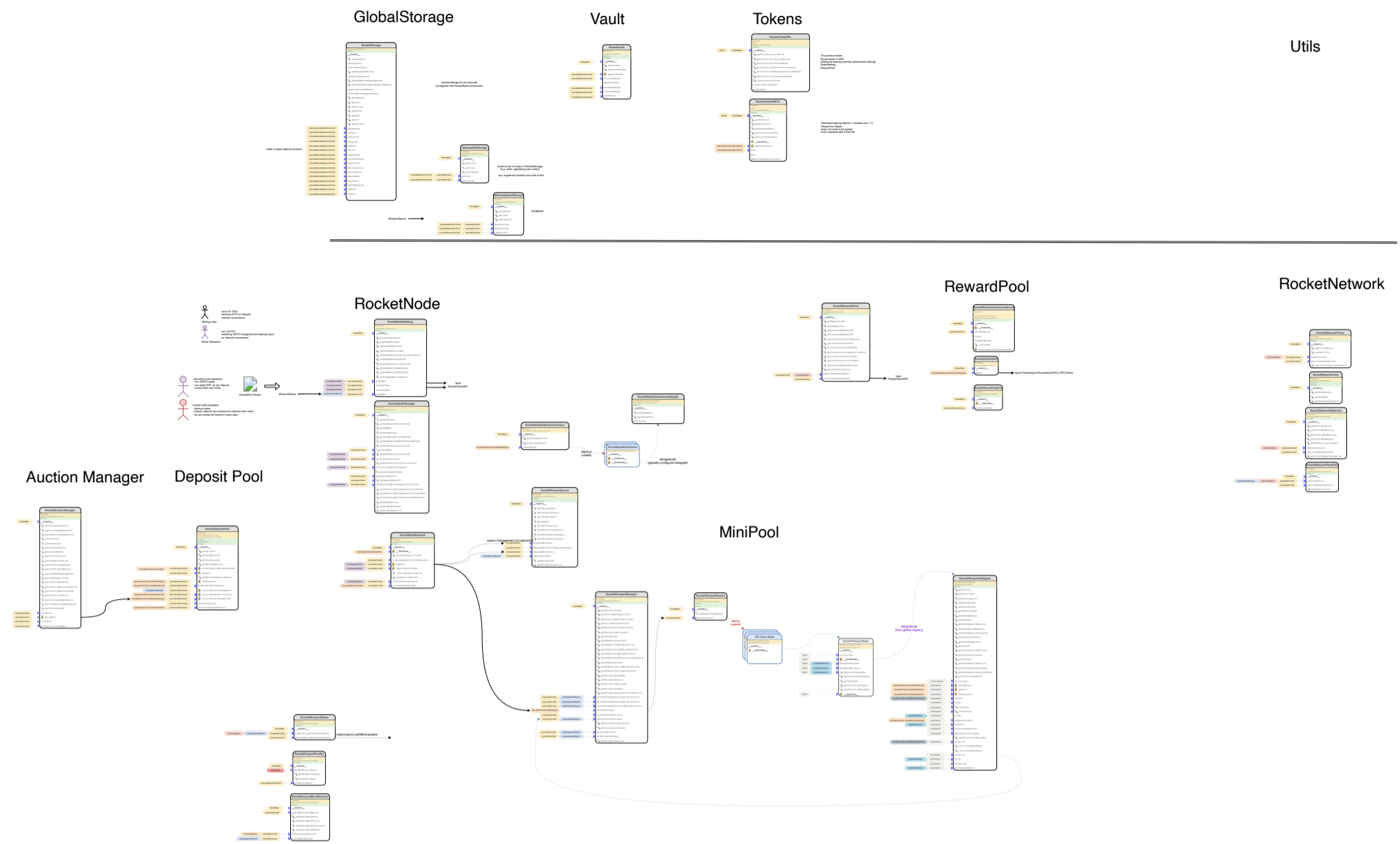
immutable Minor

Acknowledged

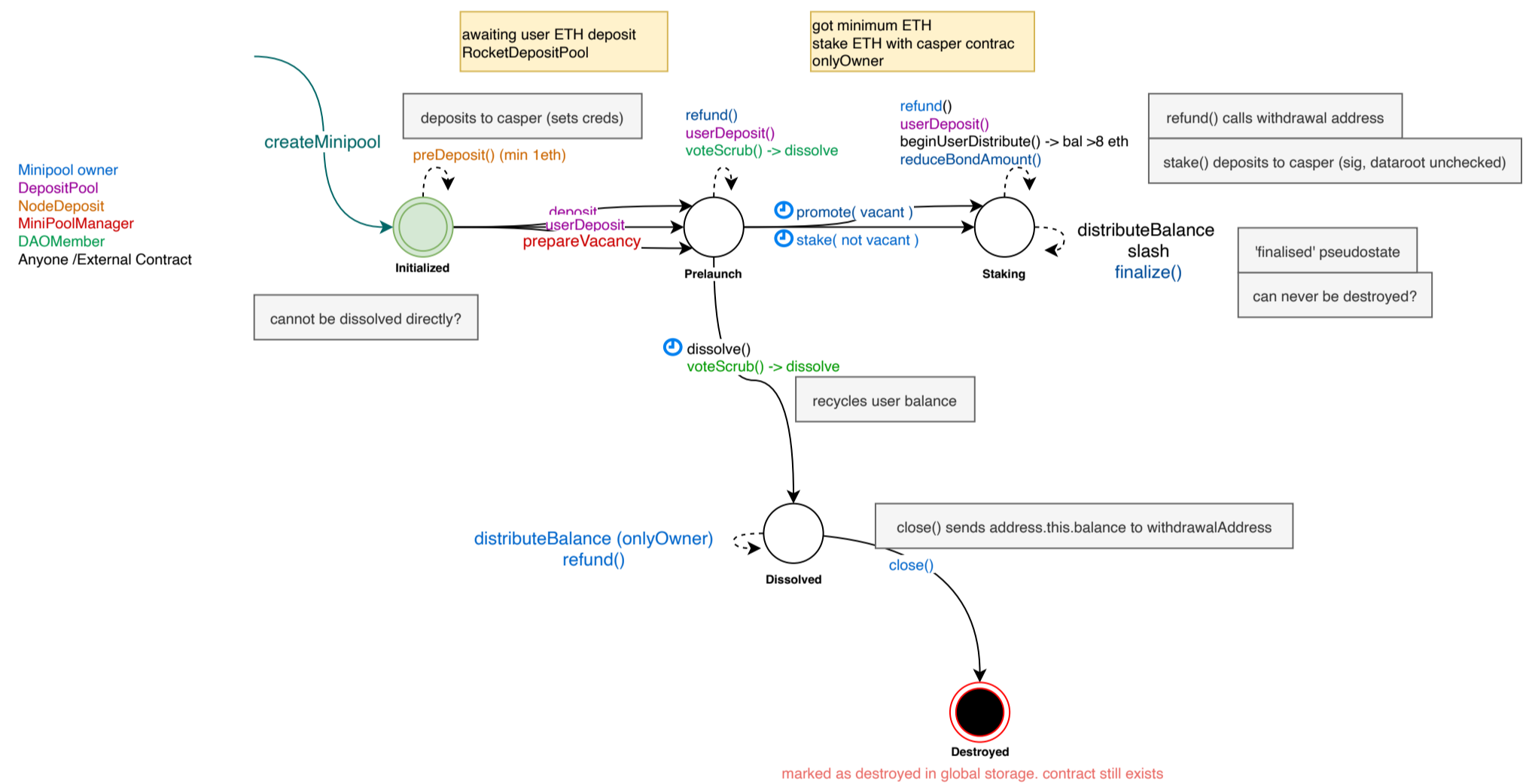
5.20 ...

This section describes the top-level/deployable contracts, their inheritance structure and interfaces, actors, permissions and important contract interactions of the **system** under review.

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The  icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that that contract is used in a usingFor declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.



RocketPool 1.2 - Architecture excluding ODAO



RocketPool 1.2 - MiniPool State Diagram

5 Findings

Each issue has an assigned severity:

- Minor** issues are subjective 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.
- Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- 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** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 RocketNodeDistributorDelegate - Reentrancy in `distribute()` allows node owner to drain distributor funds **Critical** ✓ Fixed

Resolution

Fixed in <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> by implementing a custom reentrancy guard via a new state variable `lock` that is appended to the end of the storage layout. The reentrancy guard is functionally equivalent to the OpenZeppelin implementation. The method was not refactored to give user funds priority over the node share. Additionally, the client provided the following statement:

We acknowledge this as a critical issue and have solved with a reentrancy guard.

We followed OpenZeppelin's design for a reentrancy guard. We were unable to use it directly as it is hardcoded to use storage slot 0 and because we already have deployment of this delegate in the wild already using storage slot 0 for another purpose, we had to append it to the end of the existing storage layout.

Description

The `distribute()` function distributes the contract's balance between the node operator and the user. The node operator is returned their initial collateral, including a fee. The rest is returned to the RETH token contract as user collateral.

After determining the node owner's share, the contract transfers `ETH` to the node withdrawal address, which can be the configured withdrawal address or the node address. Both addresses may potentially be a malicious contract that recursively calls back into the `distribute()` function to retrieve the node share multiple times until all funds are drained from the contract. The `distribute()` function is not protected against reentrancy:

code/contracts/contract/node/RocketNodeDistributorDelegate.sol:L59-L73

```
/// @notice Distributes the balance of this contract to its owners
function distribute() override external {
    // Calculate node share
    uint256 nodeShare = getNodeShare();
    // Transfer node share
    address withdrawalAddress = rocketStorage.getNodeWithdrawalAddress(nodeAddress);
    (bool success,) = withdrawalAddress.call{value : nodeShare}("");
    require(success);
    // Transfer user share
    uint256 userShare = address(this).balance;
    address rocketTokenRETH = rocketStorage.getAddress(rocketTokenRETHKey);
    payable(rocketTokenRETH).transfer(userShare);
    // Emit event
    emit FeesDistributed(nodeAddress, userShare, nodeShare, block.timestamp);
}
```

We also noticed that any address could set a withdrawal address as there is no check for the caller to be a registered node. In fact, the caller can be the withdrawal address or node operator.

code/contracts/contract/RocketStorage.sol:L118-L133

```
// Set a node's withdrawal address
function setWithdrawalAddress(address _nodeAddress, address _newWithdrawalAddress, bool _confirm) external override {
    // Check new withdrawal address
    require(_newWithdrawalAddress != address(0x0), "Invalid withdrawal address");
    // Confirm the transaction is from the node's current withdrawal address
    address withdrawalAddress = getNodeWithdrawalAddress(_nodeAddress);
    require(withdrawalAddress == msg.sender, "Only a tx from a node's withdrawal address can update it");
    // Update immediately if confirmed
    if (_confirm) {
        updateWithdrawalAddress(_nodeAddress, _newWithdrawalAddress);
    }
    // Set pending withdrawal address if not confirmed
    else {
        pendingWithdrawalAddresses[_nodeAddress] = _newWithdrawalAddress;
    }
}
```

Recommendation

Add a reentrancy guard to functions that interact with untrusted contracts. Adhere to the checks-effects pattern and send user funds to the 'trusted' RETH contract first. Only then send funds to the node's withdrawal address.

5.2 RocketMinipoolDelegateOld - Node operator may reenter `finalise()` to manipulate accounting Major ✓ Fixed

Resolution

The client acknowledges the finding and provided the following statement:

We are aware of this issue (it was reported via our Immunefi bug bounty). It is live but that code path is inaccessible. It requires the oDAO to mark a minipool as Withdrawable which we don't do and have removed from the withdrawal process moving forward.

In a later revision, the development team fixed the issue in the following commit:

[73d5792a671db5d2f4dcbd35737e729f9e01aa11](https://github.com/rocket-pool/rocket-pool/commit/73d5792a671db5d2f4dcbd35737e729f9e01aa11)

Description

In the old Minipool delegate contract, a node operator may call the `finalise()` function to finalize a Minipool. As part of this process, a call to `_refund()` may be performed if there is a node refund balance to be transferred. This will send an amount of `nodeRefundBalance` in ETH to the `nodeWithdrawalAddress` via a low-level call, handing over control flow to an - in terms of the system - untrusted external account that this node operator controls. The node operator, therefore, is granted to opportunity to call back into `finalise()`, which is not protected against reentrancy and violates the checks-effects-interactions pattern (`finalised = true` is only set at the very end), to manipulate the following system settings:

- `node.minipools.finalised.count<NodeAddress>`: NodeAddress finalised count increased twice instead

- `minipools.finalised.count` : global finalised count increased twice
- `eth.matched.node.amount<NodeAddress>` - NodeAddress eth matched amount potentially reduced too many times; has an impact on `getNodeETHCollateralisationRatio -> GetNodeShare`, `getNodeETHProvided -> getNodeEffectiveRPLStake` and `getNodeETHProvided->getNodeMaximumRPLStake->withdrawRPL` and is the limiting factor when withdrawing RPL to ensure the pools stay collateralized.

Note: `RocketMinipoolDelegateOld` is assumed to be the currently deployed MiniPool implementation. Users may upgrade from this delegate to the new version and can roll back at any time and re-upgrade, even within the same transaction (see [issue 5.3](#)).

The following is an annotated call stack from a node operator calling `minipool.finalise()` reentering `finalise()` once more on their Minipool:

```
finalise() -->
status == MinipoolStatus.Withdrawable //<-- true
withdrawalBlock > 0 //<-- true
_finalise() -->
!finalised //<-- true
_refund()
nodeRefundBalance = 0 //<-- reset refund balance
----> extCall: nodeWithdrawalAddress
----> reenter: finalise()
status == MinipoolStatus.Withdrawable //<-- true
withdrawalBlock > 0 //<-- true
_finalise() -->
!finalised //<-- true
nodeRefundBalance > 0 //<-- false; no refund()
address(this).balance to RETH
RocketTokenRETHInterface(rocketTokenRETH).depositExcessCollateral()
rocketMinipoolManager.incrementNodeFinalisedMinipoolCount(nodeAddress) //<-- 1st time
eventually call rocketDAONodeTrusted.decrementMemberUnbondedValidatorCount(nodeAddress);
finalised = true;
<--- return from reentrant call
<--- return from _refund()
address(this).balance to RETH //<-- NOP as balance was sent to RETH already
RocketTokenRETHInterface(rocketTokenRETH).depositExcessCollateral(); //<-- does not revert
rocketMinipoolManager.incrementNodeFinalisedMinipoolCount(nodeAddress); //<-- no revert, increases
'node.minipools.finalised.count', 'minipools.finalised.count', reduces 'eth.matched.node.amount' one to
many times
eventually call rocketDAONodeTrusted.decrementMemberUnbondedValidatorCount(nodeAddress); //<-- manipulates
'member.validator.unbonded.count' by +1
finalised = true; //<-- is already 'true', gracefully continues
<--- returns
```

code/contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol:L182-L191

```
// Called by node operator to finalise the pool and unlock their RPL stake
function finalise() external override onlyInitialised onlyMinipoolOwnerOrWithdrawalAddress(msg.sender) {
    // Can only call if withdrawable and can only be called once
    require(status == MinipoolStatus.Withdrawable, "Minipool must be withdrawable");
    // Node operator cannot finalise the pool unless distributeBalance has been called
    require(withdrawalBlock > 0, "Minipool balance must have been distributed at least once");
    // Finalise the pool
    _finalise();
}
```

`_refund()` handing over control flow to `nodeWithdrawalAddress`

code/contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol:L311-L341

```
// Perform any slashings, refunds, and unlock NO's stake
function _finalise() private {
    // Get contracts
    RocketMinipoolManagerInterface rocketMinipoolManager = RocketMinipoolManagerInterface(getContractAddress("rocketMinipoolMana
    // Can only finalise the pool once
    require(!finalised, "Minipool has already been finalised");
    // If slash is required then perform it
    if (nodeSlashBalance > 0) {
        _slash();
    }
    // Refund node operator if required
    if (nodeRefundBalance > 0) {
        _refund();
    }
    // Send any left over ETH to rETH contract
    if (address(this).balance > 0) {
        // Send user amount to rETH contract
        payable(rocketTokenRETH).transfer(address(this).balance);
    }
    // Trigger a deposit of excess collateral from rETH contract to deposit pool
    RocketTokenRETHInterface(rocketTokenRETH).depositExcessCollateral();
    // Unlock node operator's RPL
    rocketMinipoolManager.incrementNodeFinalisedMinipoolCount(nodeAddress);
    // Update unbonded validator count if minipool is unbonded
    if (depositType == MinipoolDeposit.Empty) {
        RocketDAONodeTrustedInterface rocketDAONodeTrusted = RocketDAONodeTrustedInterface(getContractAddress("rocketDAONodeTrus
        rocketDAONodeTrusted.decrementMemberUnbondedValidatorCount(nodeAddress);
    }
    // Set finalised flag
    finalised = true;
}
```

code/contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol:L517-L528

```

function _refund() private {
    // Update refund balance
    uint256 refundAmount = nodeRefundBalance;
    nodeRefundBalance = 0;
    // Get node withdrawal address
    address nodeWithdrawalAddress = rocketStorage.getNodeWithdrawalAddress(nodeAddress);
    // Transfer refund amount
    (bool success,) = nodeWithdrawalAddress.call{value : refundAmount}("");
    require(success, "ETH refund amount was not successfully transferred to node operator");
    // Emit ether withdrawn event
    emit EtherWithdrawn(nodeWithdrawalAddress, refundAmount, block.timestamp);
}

```

Methods adjusting system settings called twice:

code/contracts/contract/old/minipool/RocketMinipoolManagerOld.sol:L265-L272

```

// Increments _nodeAddress' number of minipools that have been finalised
function incrementNodeFinalisedMinipoolCount(address _nodeAddress) override external onlyLatestContract("rocketMinipoolManager",
    // Update the node specific count
    addUint(keccak256(abi.encodePacked("node.minipools.finalised.count", _nodeAddress)), 1);
    // Update the total count
    addUint(keccak256(bytes("minipools.finalised.count")), 1);
}

```

code/contracts/contract/dao/node/RocketDAONodeTrusted.sol:L139-L142

```

}
function decrementMemberUnbondedValidatorCount(address _nodeAddress) override external onlyLatestContract("rocketDAONodeTrusted"
    subUint(keccak256(abi.encodePacked(daoNameSpace, "member.validator.unbonded.count", _nodeAddress)), 1);
}

```

Recommendation

We recommend setting the `finalised = true` flag immediately after checking for it. Additionally, the function flow should adhere to the checks-effects-interactions pattern whenever possible. We recommend adding generic reentrancy protection whenever the control flow is handed to an untrusted entity.

5.3 RocketMinipoolDelegate - Sandwiching of Minipool calls can have unintended side effects Major

Resolution

The client provided the following statement:

The slashed value is purely for NO informational purposes and not used in any logic in the contracts so this example is benign as you say. We have fixed this particular issue by moving the slashed boolean out of the delegate and into RocketMinipoolLManager. It is now set on any call to rocketNodeStaking.slashRPL which covers both old delegate and new.

We appreciate that the finding was more a classification of potential issues with upgrades and rollbacks. At this stage, we cannot change this functionality as it is already deployed in a non-upgradable way to over 12,000 contracts.

As this is more of a guidance and there is no immediate threat, we don't believe this should be considered a 'major' finding.

With <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> the `slashed` flag was moved to `RocketNodeStaking.slashRPL()` (`minipool.rpl.slashed|<msg.sender> = true`).

The audit team acknowledges that this issue does not provide a concrete exploit that puts funds at risk. However, due to the sensitive nature and potential for issues regarding future updates, we stand by the initial severity rating as it stands for security vulnerabilities that may not be directly exploitable or require certain conditions to be exploited.

Description

The `RocketMinipoolBase` contract exposes the functions `delegateUpgrade` and `delegateRollback`, allowing the minipool owner to switch between delegate implementations. While giving the minipool owner a chance to roll back potentially malfunctioning upgrades, the fact that upgrades and rollback are instantaneous also gives them a chance to alternate between executing old and new code (e.g. by utilizing callbacks) and sandwich user calls to the minipool.

Examples

Assuming the latest minipool delegate implementation, any user can call `RocketMinipoolDelegatE.slash`, which slashes the node operator's RPL balance if a slashing has been recorded on their validator. To mark the minipool as having been slashed, the `slashed` contract variable is set to `true`. A minipool owner can avoid this flag from being set By sandwiching the user calls:

1. Minipool owner rolls back to the old implementation from `RocketMinipoolDelegatEOld.sol`
2. User calls `slash` on the now old delegate implementation (where `slashed` is not set)
3. Minipool owner upgrades to the latest delegate implementation again

In detail, the new slash implementation:

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L687-L696

```

function _slash() private {
    // Get contracts
    RocketNodeStakingInterface rocketNodeStaking = RocketNodeStakingInterface(getContractAddress("rocketNodeStaking"));
    // Slash required amount and reset storage value
    uint256 slashAmount = nodeSlashBalance;
    nodeSlashBalance = 0;
    rocketNodeStaking.slashRPL(nodeAddress, slashAmount);
    // Record slashing
    slashed = true;
}

```

Compared to the old slash implementation:

code/contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol:L531-L539

```

function _slash() private {
    // Get contracts
    RocketNodeStakingInterface rocketNodeStaking = RocketNodeStakingInterface(getContractAddress("rocketNodeStaking"));
    // Slash required amount and reset storage value
    uint256 slashAmount = nodeSlashBalance;
    nodeSlashBalance = 0;
    rocketNodeStaking.slashRPL(nodeAddress, slashAmount);
}

```

While the bypass of `slashed` being set is a benign example, the effects of this issue, in general, could result in a significant disruption of minipool operations and potentially affect the system's funds. The impact highly depends on the changes introduced by future minipool upgrades.

Recommendation

We recommend limiting upgrades and rollbacks to prevent minipool owners from switching implementations with an immediate effect. A time lock can fulfill this purpose when a minipool owner announces an upgrade to be done at a specific block. A warning can precede user-made calls that an upgrade is pending, and their interaction can have unintended side effects.

5.4 RocketDAONodeTrustedActions - No way to access ETH provided by non-member votes Major

Acknowledged

Resolution

According to the client, this is the intended behavior. The client provided the following statement:

This is by design.

Description

DAO members can challenge nodes to prove liveness for free. Non-DAO members must provide `members.challenge.cost = 1 eth` to start a challenge. However, the provided challenge cost is locked within the contract instead of being returned or recycled as system collateral.

Examples

code/contracts/contract/dao/node/RocketDAONodeTrustedActions.sol:L181-L192

```

// In the event that the majority/all of members go offline permanently and no more proposals could be passed, a current member or
// If it does not respond in the given window, it can be removed as a member. The one who removes the member after the challenge is
// This should only be used in an emergency situation to recover the DAO. Members that need removing when consensus is still viable
function actionChallengeMake(address _nodeAddress) override external onlyTrustedNode(_nodeAddress) onlyRegisteredNode(msg.sender)
    // Load contracts
    RocketDAONodeTrustedInterface rocketDAONode = RocketDAONodeTrustedInterface(getContractAddress("rocketDAONodeTrusted"));
    RocketDAONodeTrustedSettingsMembersInterface rocketDAONodeTrustedSettingsMembers = RocketDAONodeTrustedSettingsMembersInterf
    // Members can challenge other members for free, but for a regular bonded node to challenge a DAO member, requires non-refundab
    if(rocketDAONode.getMemberIsValid(msg.sender) != true) require(msg.value == rocketDAONodeTrustedSettingsMembers.getChallenge
    // Can't challenge yourself duh
    require(msg.sender != _nodeAddress, "You cannot challenge yourself");
    // Is this member already being challenged?

```

Recommendation

We recommend locking the ETH inside the contract during the challenge process. If a challenge is refuted, we recommend feeding the locked value back into the system as protocol collateral. If the challenge succeeds and the node is kicked, it is assumed that the challenger will be repaid the amount they had to lock up to prove non-liveness.

5.5 Multiple checks-effects violations Major

Resolution

The client provided the following statement:

In many of the cited examples, the "external call" is a call to another network contract that has the same privileges as the caller. Preventing reentrancy against our own internal contracts provides no additional security. If a malicious contract is introduced via a malicious oDAO they already have full keys to the kingdom.

None of the examples provide an attack surface and so we don't believe this to be a 'major' finding and should be downgraded.

This finding highlights our concerns about a dangerous pattern used throughout the codebase that may eventually lead to exploitable scenarios if continued to be followed, especially on codebases that do not employ protective measures against reentrant calls. This report also flagged one such exploitable instance, leading to a critical exploitable issue in one of the components.

This repeated occurrence led us to flag this as a major issue to highlight a general error and attack surface present in several places.

From our experience, there are predominantly positive side-effects of adhering to safe coding patterns, even for trusted contract interactions, as developers indirectly follow or pick up the coding style from existing code, reducing the likelihood of following a pattern that may be prone to be taken advantage of.

For example, to a developer, it might not always be directly evident that control flow is passed to potentially untrusted components/addresses from the code itself, especially when calling multiple 'trusted' components in the system. Furthermore, individual components down the call stack may be updated at later times, introducing an untrusted external call (i.e., because funds are refunded) and exposing the initially calling contract to a reentrancy-type issue. Therefore, we highly recommend adhering to a safe checks-effects pattern even though the contracts mainly interact with other trusted components and build secure code based on defense-in-depth principles to contain potential damage in favor of assuming worst-case scenarios.

Description

Throughout the system, there are various violations of the checks-effects-interactions pattern where the contract state is updated after an external call. Since large parts of the Rocket Pool system's smart contracts are not guarded against reentrancy, the external call's recipient may reenter and potentially perform malicious actions that can impact the overall accounting and, thus, system funds.

Examples

`distributeToOwner()` sends the contract's balance to the node or the withdrawal address **before** clearing the internal accounting:

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L564-L581

```
/// @notice Withdraw node balances from the minipool and close it. Only accepts calls from the owner
function close() override external onlyMinipoolOwner(msg.sender) onlyInitialised {
    // Check current status
    require(status == MinipoolStatus.Dissolved, "The minipool can only be closed while dissolved");
    // Distribute funds to owner
    distributeToOwner();
    // Destroy minipool
    RocketMinipoolManagerInterface rocketMinipoolManager = RocketMinipoolManagerInterface(getContractAddress("rocketMinipoolMana
    require(rocketMinipoolManager.getMinipoolExists(address(this)), "Minipool already closed");
    rocketMinipoolManager.destroyMinipool();
    // Clear state
    nodeDepositBalance = 0;
    nodeRefundBalance = 0;
    userDepositBalance = 0;
    userDepositBalanceLegacy = 0;
    userDepositAssignedTime = 0;
}
```

The withdrawal block should be set before any other contracts are called:

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L498-L499

```
// Save block to prevent multiple withdrawals within a few blocks
withdrawalBlock = block.number;
```

The `slashed` state should be set before any external calls are made:

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L686-L696

```
/// @dev Slash node operator's RPL balance based on nodeSlashBalance
function _slash() private {
    // Get contracts
    RocketNodeStakingInterface rocketNodeStaking = RocketNodeStakingInterface(getContractAddress("rocketNodeStaking"));
    // Slash required amount and reset storage value
    uint256 slashAmount = nodeSlashBalance;
    nodeSlashBalance = 0;
    rocketNodeStaking.slashRPL(nodeAddress, slashAmount);
    // Record slashing
    slashed = true;
}
```

In the bond reducer, the accounting values should be cleared before any external calls are made:

code/contracts/contract/minipool/RocketMinipoolBondReducer.sol:L120-L134

```

// Get desired to amount
uint256 newBondAmount = getUint(keccak256(abi.encodePacked("minipool.bond.reduction.value", msg.sender)));
require(rocketNodeDeposit.isValidDepositAmount(newBondAmount), "Invalid bond amount");
// Calculate difference
uint256 existingBondAmount = minipool.getNodeDepositBalance();
uint256 delta = existingBondAmount.sub(newBondAmount);
// Get node address
address nodeAddress = minipool.getNodeAddress();
// Increase ETH matched or revert if exceeds limit based on current RPL stake
rocketNodeDeposit.increaseEthMatched(nodeAddress, delta);
// Increase node operator's deposit credit
rocketNodeDeposit.increaseDepositCreditBalance(nodeAddress, delta);
// Clean up state
deleteUint(keccak256(abi.encodePacked("minipool.bond.reduction.time", msg.sender)));
deleteUint(keccak256(abi.encodePacked("minipool.bond.reduction.value", msg.sender)));

```

The counter for reward snapshot execution should be incremented before RPL gets minted:

code/contracts/contract/rewards/RocketRewardsPool.sol:L210-L213

```

// Execute inflation if required
rplContract.inflationMintTokens();
// Increment the reward index and update the claim interval timestamp
incrementRewardIndex();

```

Recommendation

We recommend following the checks-effects-interactions pattern and adjusting any contract state variables before making external calls. With the upgradeable nature of the system, we also recommend strictly adhering to this practice when all external calls are being made to trusted network contracts.

5.6 Minipool state machine design and pseudo-states Medium Acknowledged

Resolution

The client acknowledges the finding and provided the following statement.

We agree that the state machine is complicated. This is a symptom of technical debt and backwards compatibility.

There is no actionable response to this finding as we cannot make changes to the existing 12,000 contracts already deployed.

We want to emphasize that this finding strongly suggests that there are design deficits in the minipool state machine that, sooner or later, may impact the overall system's security. We suggest refactoring a clean design with clear transitions and states for the current iteration removing technical debt from future versions. This may mean that it may be warranted to release a new major Rocketpool version as a standalone system with a clean migration path avoiding potential problems otherwise introduced by dealing with the current technical debt.

Description

The development team has provided the assessment team with a Minipool state machine diagram. In this document, the `Destroyed` and `Finalised` states are denoted as fully qualified Minipool states. However, these conditions are pseudo-states. Specifically, the `Destroyed` pseudo-state leaves the Minipool in the actual `Dissolved` state and removes it from the Minipool accounting components. The `Finalised` pseudo-state sets the `finalised` flag on the Minipool without changing its original state. Actors may still be able to execute functions on the Minipool while it should be in an end state.

Recommendation

We strongly discourage the use of pseudo-states in state machines as they make the state machine less intuitive and present challenges in mapping state transitions to the code base. Real states and transitions should be used where possible.

Generally, we recommend the following when designing state machines:

- Using clear and descriptive transition names,
- Avoiding having multiple transitions with the same trigger,
- Modeling decisions in the form of state transitions rather than states themselves.

In any case, every Minipool should terminate in a clear end state.

5.7 RocketMinipoolDelegate - Redundant `refund()` call on forced finalization Medium ✓ Fixed

Resolution

Fixed in <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> by refactoring `refund()` to avoid a double invocation of `_refund()` in the `_finalise()` codepath.

Fixed per the recommendation. Thanks.

Description

The `RocketMinipoolDelegate.refund` function will force finalization if a user previously distributed the pool. However, `_finalise` already calls `_refund()` if there is a node refund balance to transfer, making the additional call to `_refund()` in `refund()` obsolete.

Examples

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L200-L209

```
function refund() override external onlyMinipoolOwnerOrWithdrawalAddress(msg.sender) onlyInitialised {
    // Check refund balance
    require(nodeRefundBalance > 0, "No amount of the node deposit is available for refund");
    // If this minipool was distributed by a user, force finalisation on the node operator
    if (!finalised && userDistributed) {
        _finalise();
    }
    // Refund node
    _refund();
}
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L445-L459

```
function _finalise() private {
    // Get contracts
    RocketMinipoolManagerInterface rocketMinipoolManager = RocketMinipoolManagerInterface(getContractAddress("rocketMinipoolMana
    // Can only finalise the pool once
    require(!finalised, "Minipool has already been finalised");
    // Set finalised flag
    finalised = true;
    // If slash is required then perform it
    if (nodeSlashBalance > 0) {
        _slash();
    }
    // Refund node operator if required
    if (nodeRefundBalance > 0) {
        _refund();
    }
}
```

Recommendation

We recommend refactoring the if condition to contain `_refund()` in the else branch.

5.8 Sparse documentation and accounting complexity Medium Acknowledged

Resolution

The client acknowledges the finding and provided the following statement:

Acknowledged and agree.

Description

Throughout the project, inline documentation is either sparse or missing altogether. Furthermore, few technical documents about the system's design rationale are available. The recent releases' increased complexity makes it significantly harder to trace the flow of funds through the system as components change semantics, are split into separate contracts, etc.

It is essential that documentation not only outlines what is being done but also *why* and what a function's role in the system's "bigger picture" is. Many comments in the code base fail to fulfill this requirement and are thus redundant, e.g.

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L292-L293

```
// Sanity check that refund balance is zero
require(nodeRefundBalance == 0, "Refund balance not zero");
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L333-L334

```
// Remove from vacant set
rocketMinipoolManager.removeVacantMinipool();
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L381-L383

```
if (ownerCalling) {
    // Finalise the minipool if the owner is calling
    _finalise();
}
```

The increased complexity and lack of documentation can increase the likelihood of developer error. Furthermore, the time spent maintaining the code and introducing new developers to the code base will drastically increase. This effect can be especially problematic in the system's accounting of funds as the various stages of a Minipool imply different flows of funds and interactions with external dependencies. Documentation should explain the rationale behind specific hardcoded values, such as the magic `8 ether` boundary for withdrawal detection. An example of a lack of documentation and distribution across components is the calculation and influence of `ethMatched` as it plays a role in:

- the minipool bond reducer,
- the node deposit contract,
- the node manager, and

- the node staking contract.

Recommendation

As the Rocketpool system grows in complexity, we highly recommend significantly increasing the number of inline comments and general technical documentation and exploring ways to centralize the system's accounting further to provide a clear picture of which funds move where and at what point in time. Where the flow of funds is obscured because multiple components or multi-step processes are involved, we recommend adding extensive inline documentation to give context.

5.9 RocketNodeDistributor - Missing `extcodesize` check in dynamic proxy Medium Won't Fix

Resolution

The client decided not to address the finding with the upcoming update. As per their assessment, the scenario outlined would require a series of misconfigurations/failures and hence is unlikely to happen. Following a defense-in-depth approach we, nevertheless, urge to implement safeguards on multiple layers as a condition like this can easily go undetected. However, after reviewing the feedback provided by the client we share the assessment that the finding should be downgraded from Major to Medium as funds are not at immediate risk and they can recover from this problem by fixing the delegate. For transparency, the client provided the following statement:

Agree that an `extcodesize` check here would add safety against a future mistake. But it does require a failure at many points for it to actually lead to an issue. Because this contract is not getting upgraded in Atlas, we will leave it as is. We will make note to add a safety check on it in a future update of this contract.

We don't believe this constitutes a 'major' finding given that it requires a future significant failure. If such a failure were to happen, the impact is also minimal as any calls to `distribute()` would simply do nothing. A contract upgrade would fix the problem and no funds would be at risk.

Description

`RocketNodeDistributor` dynamically retrieves the currently set delegate from the centralized `RocketStorage` contract. The target contract (delegate) is resolved inside the fallback function. It may return `address(0)`. `rocketStorage.getAddress()` does not enforce that the requested settings key exists, which may lead to `RocketNodeDistributor` delegate-calling into `address(0)`, which returns no error. This might stay undetected when calling `RocketNodeDistributorDelegate.distribute()` as the method does not return a value, which is consistent with calling a target address with no code.

Examples

`code/contracts/contract/node/RocketNodeDistributor.sol:L23-L31`

```
fallback() external payable {
    address _target = rocketStorage.getAddress(distributorStorageKey);
    assembly {
        calldatacopy(0x0, 0x0, calldatasize())
        let result := delegatecall(gas(), _target, 0x0, calldatasize(), 0x0, 0)
        returndatacopy(0x0, 0x0, returndatasize())
        switch result case 0 {revert(0, returndatasize())} default {return (0, returndatasize())}
    }
}
```

`code/contracts/contract/RocketStorage.sol:L153-L155`

```
function getAddress(bytes32 _key) override external view returns (address r) {
    return addressStorage[_key];
}
```

Recommendation

Before delegate-calling into the target contract, check if it exists.

```
assembly {
    codeSize := extcodesize(_target)
}
require(codeSize > 0);
```

5.10 Kicked oDAO members' votes taken into account Medium Acknowledged

Resolution

The client acknowledges the finding and provided the following statement:

We are aware of this limitation but the additional changes required to implement a fix outweigh the concern in our opinion.

Description

oDAO members can vote on proposals or submit external data to the system, acting as an oracle. Data submission is based on a vote by itself, and multiple oDAO members must submit the same data until a configurable threshold (51% by default) is reached

for the data to be confirmed.

When a member gets kicked or leaves the oDAO after voting, their vote is still accounted for while the total number of oDAO members decreases.

A (group of) malicious oDAO actors may exploit this fact to artificially lower the consensus threshold by voting for a proposal and then leaving the oDAO. This will leave excess votes with the proposal while the total member count decreases.

For example, let's assume there are 17 oDAO members. 9 members must vote for the proposal for it to pass (52.9%). Let's assume 8 members voted for, and the rest abstained and is against the proposal (47%, threshold not met). The proposal is unlikely to pass unless two malicious oDAO members leave the DAO, lowering the member count to 15 in an attempt to manipulate the vote, suddenly inflating vote power from 8/17 (47%; rejected) to 8/15 (53.3%; passed).

The crux is that the votes of ex-oDAO members still count, while the quorum is based on the current oDAO member number.

Here are some examples, however, this is a general pattern used for oDAO votes in the system.

Example: RocketNetworkPrices

Members submit votes via `submitPrices()`. If the threshold is reached, the proposal is executed. Quorum is based on the current oDAO member count, votes of ex-oDAO members are still accounted for. If a proposal is a near miss, malicious actors can force execute it by leaving the oDAO, lowering the threshold, and then calling `executeUpdatePrices()` to execute it.

code/contracts/contract/network/RocketNetworkPrices.sol:L75-L79

```
RocketDAONodeTrustedInterface rocketDAONodeTrusted = RocketDAONodeTrustedInterface(getContractAddress("rocketDAONodeTrusted"));
if (calcBase.mul(submissionCount).div(rocketDAONodeTrusted.getMemberCount()) >= rocketDAOProtocolSettingsNetwork.getNodeConsensus
    // Update the price
    updatePrices(_block, _rplPrice);
}
```

code/contracts/contract/network/RocketNetworkPrices.sol:L85-L86

```
function executeUpdatePrices(uint256 _block, uint256 _rplPrice) override external onlyLatestContract("rocketNetworkPrices", addr
    // Check settings
```

RocketMinipoolBondReducer

The `RocketMinipoolBondReducer` contract's `voteCancelReduction` function takes old votes of previously kicked oDAO members into account. This results in the vote being significantly higher and increases the potential for malicious actors, even after their removal, to sway the vote. Note that a canceled bond reduction cannot be undone.

code/contracts/contract/minipool/RocketMinipoolBondReducer.sol:L94-L98

```
RocketDAONodeTrustedSettingsMinipoolInterface rocketDAONodeTrustedSettingsMinipool = RocketDAONodeTrustedSettingsMinipoolInterfa
uint256 quorum = rocketDAONode.getMemberCount().mul(rocketDAONodeTrustedSettingsMinipool.getCancelBondReductionQuorum()).div(cal
bytes32 totalCancelVotesKey = keccak256(abi.encodePacked("minipool.bond.reduction.vote.count", _minipoolAddress));
uint256 totalCancelVotes = getUint(totalCancelVotesKey).add(1);
if (totalCancelVotes > quorum) {
```

RocketNetworkPenalties

code/contracts/contract/network/RocketNetworkPenalties.sol:L47-L51

```
RocketDAONodeTrustedInterface rocketDAONodeTrusted = RocketDAONodeTrustedInterface(getContractAddress("rocketDAONodeTrusted"));
if (calcBase.mul(submissionCount).div(rocketDAONodeTrusted.getMemberCount()) >= rocketDAOProtocolSettingsNetwork.getNodePenaltyT
    setBool(executedKey, true);
    incrementMinipoolPenaltyCount(_minipoolAddress);
}
```

code/contracts/contract/network/RocketNetworkPenalties.sol:L54-L58

```
// Executes incrementMinipoolPenaltyCount if consensus threshold is reached
function executeUpdatePenalty(address _minipoolAddress, uint256 _block) override external onlyLatestContract("rocketNetworkPenal
    // Get contracts
    RocketDAOProtocolSettingsNetworkInterface rocketDAOProtocolSettingsNetwork = RocketDAOProtocolSettingsNetworkInterface(getCc
    // Get submission keys
```

Recommendation

Track oDAO members' votes and remove them from the tally when the removal from the oDAO is executed.

5.11 RocketDAOProtocolSettingsRewards - settings key collision Medium Acknowledged

Resolution

The client acknowledges the finding and provided the following statement:

We are aware of this limitation but making this change now with an existing deployment outweighs the concern in our opinion.

Description

A malicious user may craft a DAO protocol proposal to set a rewards claimer for a specific contract, thus overwriting another contract's settings. This issue arises due to lax requirements when choosing safe settings keys.

code/contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsRewards.sol:L36-L49

```
function setSettingRewardsClaimer(string memory _contractName, uint256 _perc) override public onlyDAOProtocolProposal {
    // Get the total perc set, can't be more than 100
    uint256 percTotal = getRewardsClaimersPercTotal();
    // If this group already exists, it will update the perc
    uint256 percTotalUpdate = percTotal.add(_perc).sub(getRewardsClaimerPerc(_contractName));
    // Can't be more than a total claim amount of 100%
    require(percTotalUpdate <= 1 ether, "Claimers cannot total more than 100%");
    // Update the total
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.totalPerc")), percTotalUpdate);
    // Update/Add the claimer amount
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.amount", _contractName)), _perc);
    // Set the time it was updated at
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.amount.updated.time", _contractName)), block.t
}
}
```

The method updates the rewards claimer for a specific contract by writing to the following two setting keys:

- `settingNameSpace.rewards.claimsgroup.amount<_contractName>`
- `settingNameSpace.rewards.claimsgroup.amount.updated.time<_contractName>`

Due to the way the settings hierarchy was chosen in this case, a malicious proposal might define a `<_contractName> = .updated.time<targetContract>` that overwrites the settings of a different contract with an invalid value.

Note that the issue of delimiter consistency is also discussed in [issue 5.12](#).

The severity rating is based on the fact that this should be detectable by DAO members. However, following a defense-in-depth approach means that such collisions should be avoided wherever possible.

Recommendation

We recommend enforcing a unique prefix and delimiter when concatenating user-provided input to setting keys. In this specific case, the settings could be renamed as follows:

- `settingNameSpace.rewards.claimsgroup.amount.value<_contractName>`
- `settingNameSpace.rewards.claimsgroup.amount.updated.time<_contractName>`

5.12 RocketDAOProtocolSettingsRewards - missing setting delimiters Medium Acknowledged

Resolution

The client acknowledges the finding and provided the following statement:

We are aware of this limitation but making this change now with an existing deployment outweighs the concern in our opinion.

Description

Settings in the Rocket Pool system are hierarchical, and namespaces are prefixed using dot delimiters.

Calling `abi.encodePacked(<string>, <string>)` on strings performs a simple concatenation. According to the settings' naming scheme, it is suggested that the following example writes to a key named: `<settingNameSpace>.rewards.claims.group.amount.<_contractName>`. However, due to missing delimiters, the actual key written to is: `<settingNameSpace>.rewards.claimsgroup.amount<_contractName>`.

Note that there is no delimiter between `claims|group` and `amount|<_contractName>`.

code/contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsRewards.sol:L36-L49

```
function setSettingRewardsClaimer(string memory _contractName, uint256 _perc) override public onlyDAOProtocolProposal {
    // Get the total perc set, can't be more than 100
    uint256 percTotal = getRewardsClaimersPercTotal();
    // If this group already exists, it will update the perc
    uint256 percTotalUpdate = percTotal.add(_perc).sub(getRewardsClaimerPerc(_contractName));
    // Can't be more than a total claim amount of 100%
    require(percTotalUpdate <= 1 ether, "Claimers cannot total more than 100%");
    // Update the total
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.totalPerc")), percTotalUpdate);
    // Update/Add the claimer amount
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.amount", _contractName)), _perc);
    // Set the time it was updated at
    setUint(keccak256(abi.encodePacked(settingNameSpace, "rewards.claims", "group.amount.updated.time", _contractName)), block.t
}
}
```

Recommendation

We recommend adding the missing intermediate delimiters. The system should enforce delimiters after the last setting key before user input is concatenated to reduce the risk of accidental namespace collisions.

5.13 Use of `address` instead of specific contract types Minor Acknowledged

Resolution

The client acknowledges the finding, removed the unnecessary casts from `canReduceBondAmount` and `voteCancelReduction` with <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6>, and provided the following statement:

Acknowledged. We will migrate to this pattern as we upgrade contracts.

Description

Rather than using a low-level address type and then casting to the safer contract type, it's better to use the best type available by default so the compiler can eventually check for type safety and contract existence and only downcast to less secure low-level types (`address`) when necessary.

Examples

`RocketStorageInterface _rocketStorage` should be declared in the arguments, removing the need to cast the address explicitly.

code/contracts/contract/minipool/RocketMinipoolBase.sol:L39-L47

```
/// @notice Sets up starting delegate contract and then delegates initialisation to it
function initialise(address _rocketStorage, address _nodeAddress) external override notSelf {
    // Check input
    require(_nodeAddress != address(0), "Invalid node address");
    require(storageState == StorageState.Undefined, "Already initialised");
    // Set storage state to uninitialised
    storageState = StorageState.Uninitialised;
    // Set rocketStorage
    rocketStorage = RocketStorageInterface(_rocketStorage);
}
```

`RocketMinipoolInterface _minipoolAddress` should be declared in the arguments, removing the need to cast the address explicitly. Downcast to low-level address if needed. The event can be redeclared with the contract type.

code/contracts/contract/minipool/RocketMinipoolBondReducer.sol:L33-L34

```
function beginReduceBondAmount(address _minipoolAddress, uint256 _newBondAmount) override external onlyLatestContract("rocketMinipool") {
    RocketMinipoolInterface minipool = RocketMinipoolInterface(_minipoolAddress);
}
```

code/contracts/contract/minipool/RocketMinipoolBondReducer.sol:L69-L76

```
/// @notice Returns whether owner of given minipool can reduce bond amount given the waiting period constraint
/// @param _minipoolAddress Address of the minipool
function canReduceBondAmount(address _minipoolAddress) override public view returns (bool) {
    RocketMinipoolInterface minipool = RocketMinipoolInterface(_minipoolAddress);
    RocketDA0NodeTrustedSettingsMinipoolInterface rocketDA0NodeTrustedSettingsMinipool = RocketDA0NodeTrustedSettingsMinipoolInterface(_minipoolAddress);
    uint256 reduceBondTime = getUint(keccak256(abi.encodePacked("minipool.bond.reduction.time", _minipoolAddress)));
    return rocketDA0NodeTrustedSettingsMinipool.isWithinBondReductionWindow(block.timestamp.sub(reduceBondTime));
}
```

code/contracts/contract/minipool/RocketMinipoolBondReducer.sol:L80-L84

```
function voteCancelReduction(address _minipoolAddress) override external onlyTrustedNode(msg.sender) onlyLatestContract("rocketMinipool") {
    // Prevent calling if consensus has already been reached
    require(!getReduceBondCancelled(_minipoolAddress), "Already cancelled");
    // Get contracts
    RocketMinipoolInterface minipool = RocketMinipoolInterface(_minipoolAddress);
}
```

Note that `abi.encode*(contractType)` assumes `address` for contract types by default. An explicit downcast is not required.

```
» Test example = Test(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)
» abi.encodePacked("hi", example)
0x68695b38da6a701c568545dcfcb03fcb875f56beddc4
» abi.encodePacked("hi", address(example))
0x68695b38da6a701c568545dcfcb03fcb875f56beddc4
```

More examples of `address _minipool` declarations:

code/contracts/contract/minipool/RocketMinipoolManager.sol:L449-L455

```
/// @dev Internal logic to set a minipool's pubkey
/// @param _pubkey The pubkey to set for the calling minipool
function _setMinipoolPubkey(address _minipool, bytes calldata _pubkey) private {
    // Load contracts
    AddressSetStorageInterface addressSetStorage = AddressSetStorageInterface(getContractAddress("addressSetStorage"));
    // Initialize minipool & get properties
    RocketMinipoolInterface minipool = RocketMinipoolInterface(_minipool);
}
```

code/contracts/contract/minipool/RocketMinipoolManager.sol:L474-L478

```
function getMinipoolDetails(address _minipoolAddress) override external view returns (MinipoolDetails memory) {
    // Get contracts
    RocketMinipoolInterface minipoolInterface = RocketMinipoolInterface(_minipoolAddress);
    RocketMinipoolBase minipool = RocketMinipoolBase(payable(_minipoolAddress));
    RocketNetworkPenaltiesInterface rocketNetworkPenalties = RocketNetworkPenaltiesInterface(getContractAddress("rocketNetworkPenalties"));
}
```

More examples of `RocketStorageInterface _rocketStorage` casts:

code/contracts/contract/node/RocketNodeDistributor.sol:L8-L13

```
contract RocketNodeDistributor is RocketNodeDistributorStorageLayout {
    bytes32 immutable distributorStorageKey;

    constructor(address _nodeAddress, address _rocketStorage) {
        rocketStorage = RocketStorageInterface(_rocketStorage);
        nodeAddress = _nodeAddress;
    }
}
```

Recommendation

We recommend using more specific types instead of `address` where possible. Downcast if necessary. This goes for parameter types as well as state variable types.

5.14 Redundant double casts Minor Acknowledged

Resolution

The client acknowledges the finding and provided the following statement:

Acknowledged. These contracts are non-upgradable.

Description

`_rocketStorageAddress` is already of contract type `RocketStorageInterface`.

code/contracts/contract/RocketBase.sol:L78-L82

```
/// @dev Set the main Rocket Storage address
constructor(RocketStorageInterface _rocketStorageAddress) {
    // Update the contract address
    rocketStorage = RocketStorageInterface(_rocketStorageAddress);
}
```

`_tokenAddress` is already of contract type `ERC20Burnable`.

code/contracts/contract/RocketVault.sol:L132-L138

```
function burnToken(ERC20Burnable _tokenAddress, uint256 _amount) override external onlyLatestNetworkContract {
    // Get contract key
    bytes32 contractKey = keccak256(abi.encodePacked(getContractName(msg.sender), _tokenAddress));
    // Update balances
    tokenBalances[contractKey] = tokenBalances[contractKey].sub(_amount);
    // Get the token ERC20 instance
    ERC20Burnable tokenContract = ERC20Burnable(_tokenAddress);
}
```

`_rocketTokenRPLFixedSupplyAddress` is already of contract type `IERC20`.

code/contracts/contract/token/RocketTokenRPL.sol:L47-L51

```
constructor(RocketStorageInterface _rocketStorageAddress, IERC20 _rocketTokenRPLFixedSupplyAddress) RocketBase(_rocketStorageAddress) {
    // Version
    version = 1;
    // Set the mainnet RPL fixed supply token address
    rplFixedSupplyContract = IERC20(_rocketTokenRPLFixedSupplyAddress);
}
```

Recommendation

We recommend removing the unnecessary double casts and copies of local variables.

5.15 RocketMinipoolDelegate - Missing event in `prepareVacancy` Minor ✓ Fixed

Resolution

Fixed in <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> by emitting a new event `MinipoolVacancyPrepared`.

Agreed. Added event per recommendation. Thanks.

Description

The function `prepareVacancy` updates multiple contract state variables and should therefore emit an event.

Examples

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L286-L309


```

/// @dev Sets the bond value and vacancy flag on this minipool
/// @param _bondAmount The bond amount selected by the node operator
/// @param _currentBalance The current balance of the validator on the beaconchain (will be checked by oDAO and scrubbed if not cor
function prepareVacancy(uint256 _bondAmount, uint256 _currentBalance) override external onlyLatestContract("rocketMinipoolManage
    // Check status
    require(status == MinipoolStatus.Initialised, "Must be in initialised status");
    // Sanity check that refund balance is zero
    require(nodeRefundBalance == 0, "Refund balance not zero");
    // Check balance
    RocketDAOProtocolSettingsMinipoolInterface rocketDAOProtocolSettingsMinipool = RocketDAOProtocolSettingsMinipoolInterface(ge
    uint256 launchAmount = rocketDAOProtocolSettingsMinipool.getLaunchBalance();
    require(_currentBalance >= launchAmount, "Balance is too low");
    // Store bond amount
    nodeDepositBalance = _bondAmount;
    // Calculate user amount from launch amount
    userDepositBalance = launchAmount.sub(nodeDepositBalance);
    // Flag as vacant
    vacant = true;
    preMigrationBalance = _currentBalance;
    // Refund the node whatever rewards they have accrued prior to becoming a RP validator
    nodeRefundBalance = _currentBalance.sub(launchAmount);
    // Set status to preLaunch
    setStatus(MinipoolStatus.PreLaunch);
}

```

Recommendation

Emit the missing event.

5.16 Compiler error due to missing RocketMinipoolBaseInterface Minor ✓ Fixed

Resolution

Fixed in <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> by adding the missing interface file.

Description

The interface `RocketMinipoolBaseInterface` is missing from the code repository. Manually generating the interface and adding it to the repository fixes the error.

```

⇒ npx hardhat compile
Error HH404: File ../../interface/minipool/RocketMinipoolBaseInterface.sol, imported from contracts/contract/minipool/RocketMinipoolBase.sol, not found
For more info go to https://hardhat.org/HH404 or run Hardhat with --show-stack-traces

```

Recommendation

Add the missing source unit to the repository.

5.17 Unused Imports Minor Partially Addressed

Resolution

Addressed in <https://github.com/rocket-pool/rocketpool/tree/77d7cca65b7c0557cfda078a4fc45f9ac0cc6cc6> by removing all but the following two mentioned unused imports:

- `RocketRewardsPoolInterface`
- `RocketSmoothingPoolInterface`

Description

The following source units are imported but not referenced in the importing source unit:

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L11

```
import "../../interface/rewards/RocketSmoothingPoolInterface.sol";
```

code/contracts/contract/minipool/RocketMinipoolFactory.sol:L12-L18

```

import "../../interface/minipool/RocketMinipoolManagerInterface.sol";
import "../../interface/minipool/RocketMinipoolQueueInterface.sol";
import "../../interface/node/RocketNodeStakingInterface.sol";
import "../../interface/util/AddressSetStorageInterface.sol";
import "../../interface/node/RocketNodeManagerInterface.sol";
import "../../interface/network/RocketNetworkPricesInterface.sol";
import "../../interface/dao/protocol/settings/RocketDAOProtocolSettingsMinipoolInterface.sol";

```

code/contracts/contract/minipool/RocketMinipoolFactory.sol:L8-L10

```

import "../../types/MinipoolStatus.sol";
import "../../types/MinipoolDeposit.sol";
import "../../interface/dao/node/RocketDAONodeTrustedInterface.sol";

```

code/contracts/contract/minipool/RocketMinipoolBase.sol:L7-L8

```
import "../../../../types/MinipoolDeposit.sol";
import "../../../../types/MinipoolStatus.sol";
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L13-L14

```
import "../../../../interface/network/RocketNetworkPricesInterface.sol";
import "../../../../interface/node/RocketNodeManagerInterface.sol";
```

code/contracts/contract/node/RocketNodeManager.sol:L13

```
import "../../../../interface/rewards/claims/RocketClaimNodeInterface.sol";
```

code/contracts/contract/rewards/RocketClaimDAO.sol:L7

```
import "../../../../interface/rewards/RocketRewardsPoolInterface.sol";
```

Duplicate Import:

code/contracts/contract/minipool/RocketMinipoolFactory.sol:L19-L20

```
import "../../../../interface/dao/protocol/settings/RocketDAOProtocolSettingsNodeInterface.sol";
import "../../../../interface/dao/protocol/settings/RocketDAOProtocolSettingsNodeInterface.sol";
```

The above list is exemplary, and there are likely more occurrences across the code base.

Recommendation

We recommend checking all imports and removing unused/unreferenced and unnecessary imports.

5.18 RocketMinipool - Inconsistent access control modifier declaration `onlyMinipoolOwner` Minor

Acknowledged

Resolution

Acknowledged by the client. Not addressed within [rocket-pool/rocketpool@77d7cca](#)

Agreed. This would change a lot of contracts just for a minor improvement in readability.

Description

The access control modifier `onlyMinipoolOwner` should be renamed to `onlyMinipoolOwnerOrWithdrawalAddress` to be consistent with the actual check permitting the owner or the withdrawal address to interact with the function. This would also be consistent with other declarations in the codebase.

Example

The `onlyMinipoolOwner` modifier in `RocketMinipoolBase` is the same as `onlyMinipoolOwnerOrWithdrawalAddress` in other modules.

code/contracts/contract/minipool/RocketMinipoolBase.sol:L31-L37

```
/// @dev Only allow access from the owning node address
modifier onlyMinipoolOwner() {
    // Only the node operator can upgrade
    address withdrawalAddress = rocketStorage.getNodeWithdrawalAddress(nodeAddress);
    require(msg.sender == nodeAddress || msg.sender == withdrawalAddress, "Only the node operator can access this method");
    _;
}
```

code/contracts/contract/old/minipool/RocketMinipoolOld.sol:L21-L27

```
// Only allow access from the owning node address
modifier onlyMinipoolOwner() {
    // Only the node operator can upgrade
    address withdrawalAddress = rocketStorage.getNodeWithdrawalAddress(nodeAddress);
    require(msg.sender == nodeAddress || msg.sender == withdrawalAddress, "Only the node operator can access this method");
    _;
}
```

Other declarations:

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L97-L107

```

/// @dev Only allow access from the owning node address
modifier onlyMinipoolOwner(address _nodeAddress) {
    require(_nodeAddress == nodeAddress, "Invalid minipool owner");
    -;
}

/// @dev Only allow access from the owning node address or their withdrawal address
modifier onlyMinipoolOwnerOrWithdrawalAddress(address _nodeAddress) {
    require(_nodeAddress == nodeAddress || _nodeAddress == rocketStorage.getNodeWithdrawalAddress(nodeAddress), "Invalid minipool");
    -;
}

```

code/contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol:L82-L92

```

// Only allow access from the owning node address
modifier onlyMinipoolOwner(address _nodeAddress) {
    require(_nodeAddress == nodeAddress, "Invalid minipool owner");
    -;
}

// Only allow access from the owning node address or their withdrawal address
modifier onlyMinipoolOwnerOrWithdrawalAddress(address _nodeAddress) {
    require(_nodeAddress == nodeAddress || _nodeAddress == rocketStorage.getNodeWithdrawalAddress(nodeAddress), "Invalid minipool");
    -;
}

```

Recommendation

We recommend renaming `RocketMinipoolBase.onlyMinipoolOwner` to `RocketMinipoolBase.onlyMinipoolOwnerOrWithdrawalAddress`.

5.19 RocketDAO*Settings - `settingNameSpace` should be `immutable` Minor Acknowledged

Resolution

Acknowledged by the client. Not addressed within [rocket-pool/rocketpool@77d7cca](#)

Acknowledged. We can fix this as we upgrade the related contracts.

Description

The `settingNameSpace` in the abstract contract `RocketDAONodeTrustedSettings` is only set on contract deployment. Hence, the fields should be declared immutable to make clear that the settings namespace cannot change after construction.

Examples

- `RocketDAONodeTrustedSettings`

code/contracts/contract/dao/node/settings/RocketDAONodeTrustedSettings.sol:L13-L16

```

// The namespace for a particular group of settings
bytes32 settingNameSpace;

```

code/contracts/contract/dao/node/settings/RocketDAONodeTrustedSettings.sol:L25-L30

```

// Construct
constructor(RocketStorageInterface _rocketStorageAddress, string memory _settingNameSpace) RocketBase(_rocketStorageAddress) {
    // Apply the setting namespace
    settingNameSpace = keccak256(abi.encodePacked("dao.trustednodes.setting.", _settingNameSpace));
}

```

- `RocketDAOProtocolSettings`

code/contracts/contract/dao/protocol/settings/RocketDAOProtocolSettings.sol:L13-L14

```

// The namespace for a particular group of settings
bytes32 settingNameSpace;

```

code/contracts/contract/dao/protocol/settings/RocketDAOProtocolSettings.sol:L25-L29

```

// Construct
constructor(RocketStorageInterface _rocketStorageAddress, string memory _settingNameSpace) RocketBase(_rocketStorageAddress) {
    // Apply the setting namespace
    settingNameSpace = keccak256(abi.encodePacked("dao.protocol.setting.", _settingNameSpace));
}

```

code/contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsAuction.sol:L13-L15

```

constructor(RocketStorageInterface _rocketStorageAddress) RocketDAOProtocolSettings(_rocketStorageAddress, "auction") {
    // Set version
    version = 1;
}

```


Recommendation

We recommend using the `immutable` annotation in Solidity (see [Immutable](#)).

5.20 Inefficiencies with the `onlyMinipoolOwner` modifier Acknowledged

Resolution

Acknowledged by the client. No further actions.

Correct. This change would change every single contract we have and so the benefit does not outweigh the change.

Description

If a withdrawal address has not been set (or has been zeroed out), `rocketStorage.getNodeWithdrawalAddress(nodeAddress)` returns `nodeAddress`. This outcome leads to the modifier checking the same address twice (`msg.sender == nodeAddress || msg.sender == nodeAddress`):

code/contracts/contract/minipool/RocketMinipoolBase.sol:L31-L37

```
/// @dev Only allow access from the owning node address
modifier onlyMinipoolOwner() {
    // Only the node operator can upgrade
    address withdrawalAddress = rocketStorage.getNodeWithdrawalAddress(nodeAddress);
    require(msg.sender == nodeAddress || msg.sender == withdrawalAddress, "Only the node operator can access this method");
    _;
}
```

code/contracts/contract/RocketStorage.sol:L103-L111

```
// Get a node's withdrawal address
function getNodeWithdrawalAddress(address _nodeAddress) public override view returns (address) {
    // If no withdrawal address has been set, return the nodes address
    address withdrawalAddress = withdrawalAddresses[_nodeAddress];
    if (withdrawalAddress == address(0)) {
        return _nodeAddress;
    }
    return withdrawalAddress;
}
```

5.21 RocketNodeDeposit - Duplicate check to avoid revert ✓ Fixed

Resolution

Fixed with [rocket-pool/rocketpool@3ab7af1](#) by introducing a new method `maybeAssignDeposits()` that does not revert by default but returns a boolean instead. This way, `RocketNodeDeposit` directly call the `maybeAssignDeposits()` function, avoiding the duplicate check.

This finding does not present a security-related problem in the code base, which is why we downgrade its severity to *informational*. However, we opted to keep this recommendation present in the report since it underlines a form of technical debt where old functionality is wrapped by new functionality using a workaround.

Description

When receiving and subsequently assigning deposits, the `RocketNodeDeposit` contract's `assignDeposits` function calls `RocketDAOProtocolSettingsDeposit.getAssignDepositsEnabled` and skips the assignment of funds. This is done because the `RocketDepositPool.assignDeposits` function reverts if the setting is disabled:

code/contracts/contract/deposit/RocketDepositPool.sol:L207-L212

```
function assignDeposits() override external onlyThisLatestContract {
    // Load contracts
    RocketDAOProtocolSettingsDepositInterface rocketDAOProtocolSettingsDeposit = RocketDAOProtocolSettingsDepositInterface(getCo
    // Revert if assigning is disabled
    require(!_assignDeposits(rocketDAOProtocolSettingsDeposit), "Deposit assignments are currently disabled");
}
```

However, the underlying `_assignDeposits` function already performs a check for the setting and returns prematurely to avoid assignment.

code/contracts/contract/deposit/RocketDepositPool.sol:L217-L219

```
if (!_rocketDAOProtocolSettingsDeposit.getAssignDepositsEnabled()) {
    return false;
}
```

The `rocketDAOProtocolSettingsDeposit.getAssignDepositsEnabled()` setting is checked twice. The first occurrence is in `RocketNodeDeposit.assignDeposits` and the second one in the same flow is contained in `RocketDepositPool._assignDeposits`. The second check is performed in a reverting fashion, thus requiring the top-level check in the `RocketNodeDeposit` contract to preemptively fetch and check the setting before continuing.

Recommendation

Since Rocketpool v1.2 already aims to perform an upgrade on the `RocketDepositPool` contract, we do recommend adding a separate, non-reverting version of the `RocketDepositPool.assignDeposits` function to the code base and removing the redundant preemptive check in `RocketNodeDeposit.assignDeposits`. This will improve readability and maintainability of future versions of the code, and save gas cost on deposit assignment operations.

5.22 Inconsistent Coding Style Acknowledged

Resolution

The client provided the following statement:

Acknowledge your recommendation but we are dealing with an existing deployed codebase and if we change codestyle on only the contracts we update we will end up with a codebase with different code styles which is worse than one that is internally consistent but not consistent with best practice.

Description

Deviations from the [Solidity Style Guide](#) were identified throughout the codebase. Considering how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with the help of linter tools is recommended.

Inconsistent Function naming scheme for external and internal interfaces

Throughout the codebase, private/internal functions are generally prefixed with an underscore (`_<name>`). This allows for an easy way to see if an external party can interact with a function without having to scan the declaration line for the corresponding visibility keywords. However, this naming scheme is not enforced consistently. Many internal function names are indistinguishable from external function names. It is therefore highly recommended to implement a consistent naming scheme and prefix internal functions with an underscore (`_<name>`).

code/contracts/contract/node/RocketNodeDeposit.sol:L268-L283

```
/// @dev Reverts if vacant minipools are not enabled
function checkVacantMinipoolsEnabled() private view {
    // Get contracts
    RocketDAOProtocolSettingsNodeInterface rocketDAOProtocolSettingsNode = RocketDAOProtocolSettingsNodeInterface(getContractAdd
    // Check node settings
    require(rocketDAOProtocolSettingsNode.getVacantMinipoolsEnabled(), "Vacant minipools are currently disabled");
}

/// @dev Executes an assignDeposits call on the deposit pool
function assignDeposits() private {
    RocketDAOProtocolSettingsDepositInterface rocketDAOProtocolSettingsDeposit = RocketDAOProtocolSettingsDepositInterface(getCc
    if (rocketDAOProtocolSettingsDeposit.getAssignDepositsEnabled()) {
        RocketDepositPoolInterface rocketDepositPool = RocketDepositPoolInterface(getContractAddress("rocketDepositPool"));
        rocketDepositPool.assignDeposits();
    }
}
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L339-L345

```
/// @dev Stakes the balance of this minipool into the deposit contract to set withdrawal credentials to this contract
/// @param _validatorSignature A signature over the deposit message object
/// @param _depositDataRoot The hash tree root of the deposit data object
function preStake(bytes calldata _validatorPubkey, bytes calldata _validatorSignature, bytes32 _depositDataRoot) internal {
    // Load contracts
    DepositInterface casperDeposit = DepositInterface(getContractAddress("casperDeposit"));
    RocketMinipoolManagerInterface rocketMinipoolManager = RocketMinipoolManagerInterface(getContractAddress("rocketMinipoolMana
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L651-L654

```
/// @dev Distributes the current contract balance based on capital ratio and node fee
function distributeSkimmedRewards() internal {
    uint256 rewards = address(this).balance.sub(nodeRefundBalance);
    uint256 nodeShare = calculateNodeRewards(nodeDepositBalance, getUserDepositBalance(), rewards);
}
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L661-L663

```
/// @dev Set the minipool's current status
/// @param _status The new status
function setStatus(MinipoolStatus _status) private {
```

code/contracts/contract/node/RocketNodeDeposit.sol:L202-L206

```

/// @dev Adds a minipool to the queue
function enqueueMinipool(address _minipoolAddress) private {
    // Add minipool to queue
    RocketMinipoolQueueInterface(getContractAddress("rocketMinipoolQueue")).enqueueMinipool(_minipoolAddress);
}

```

code/contracts/contract/node/RocketNodeDeposit.sol:L208-L213

```

/// @dev Reverts if node operator has not initialised their fee distributor
function checkDistributorInitialised() private view {
    // Check node has initialised their fee distributor
    RocketNodeManagerInterface rocketNodeManager = RocketNodeManagerInterface(getContractAddress("rocketNodeManager"));
    require(rocketNodeManager.getFeeDistributorInitialised(msg.sender), "Fee distributor not initialised");
}

```

code/contracts/contract/node/RocketNodeDeposit.sol:L215-L218

```

/// @dev Creates a minipool and returns an instance of it
/// @param _salt The salt used to determine the minipools address
/// @param _expectedMinipoolAddress The expected minipool address. Reverts if not correct
function createMinipool(uint256 _salt, address _expectedMinipoolAddress) private returns (RocketMinipoolInterface) {

```

code/contracts/contract/auction/RocketAuctionManager.sol:L58-L60

```

function setLotCount(uint256 _amount) private {
    setUint(keccak256("auction.lots.count"), _amount);
}

```

Appendix 1 - Files in Scope

This audit covered the following files:

SHA-1 Hash	File
e0d054c08e868a73e78f29c5b80b6b2d9d31ac43	contracts/contract/node/RocketNodeDistributorStorageLayout.sol
ed07e8bd9a7309d3755ed76bce7c04e48628b9de	contracts/contract/node/RocketNodeDistributorFactory.sol
ef40c3420e4492bbd405d239c63b2dbb23d167e3	contracts/contract/node/RocketNodeDistributor.sol
493234d4c3fc24d598067d60e746f1257bf4bb5f	contracts/contract/node/RocketNodeStaking.sol
eb50e6e9cc2eb94cd01288521f9bc73890fc483c	contracts/contract/node/RocketNodeManager.sol
e73ed34c2c3baa463dac0947b70dbb1e5bcff99d	contracts/contract/node/RocketNodeDeposit.sol
77785b4a3196f4736e574ea5e87b018d3d3e7824	contracts/contract/node/RocketNodeDistributorDelegate.sol
8749d09f3e9e80f9676fd9d48622a850d3d3f71b	contracts/contract/upgrade/RocketUpgradeOneDotTwo.sol
cd1c38eb32ab318934d94b40531e32f1ac58261e	contracts/contract/rewards/RocketMerkleDistributorMainnet.sol
2d7fcb81de5fb404c26880d8015535c5423b8651	contracts/contract/rewards/RocketRewardsPool.sol
93d3616ebaa9053f8defdebf8568ebcdc7d6b7b6	contracts/contract/rewards/RocketSmoothingPool.sol
0242996368cabfed713cceccb2313f2a0f6fe586	contracts/contract/rewards/RocketClaimDAO.sol
b0bda868dad3c7c43ad740b24a37e2cb4f52994f	contracts/contract/RocketBase.sol
e17c6b07288ea458d101cd54bd2a230d67f8722d	contracts/contract/network/RocketNetworkBalances.sol
cdf2a398e2ff8f6a55ec7efd5c6c316e34ae4966	contracts/contract/network/RocketNetworkFees.sol
9b81afb407809332c2a3f605317cfadebf3289cb	contracts/contract/network/RocketNetworkPenalties.sol
5ce805aba1d1457a80b9c9d3b49fa2753771aeee	contracts/contract/network/RocketNetworkPrices.sol
bfb92f4e76b81ac6ed1c09e405a39394b93ec398	contracts/contract/RocketVault.sol
385c9f5e2e6eeac513c923e163e531fc18d89cbc	contracts/contract/dao/node/RocketDAONodeTrustedActions.sol
3072edef2f2e1f58be33b27d6dc36071529e1f56	contracts/contract/dao/node/settings/RocketDAONodeTrustedSettingsRewards.sol
ed47d92656f92c5b09a84a343bc071aa76d20bd2	contracts/contract/dao/node/settings/RocketDAONodeTrustedSettingsProposals.sol
a4876b026c0e03777e1121616856b57a0fe20f0c	contracts/contract/dao/node/settings/RocketDAONodeTrustedSettingsMinipool.sol
154ae3907234b7cadbd35841f8c3a3582ff4e96c	contracts/contract/dao/node/settings/RocketDAONodeTrustedSettingsMembers.sol
cc57265a125d582c3b4e4e50f2ea3f2427df8133	contracts/contract/dao/node/settings/RocketDAONodeTrustedSettings.sol
0f8e23d4ad751551638956a5e4673723ab466634	contracts/contract/dao/node/RocketDAONodeTrusted.sol
2511ff09533cc83a7daf112121ed20dad88ee8a0	contracts/contract/dao/node/RocketDAONodeTrustedUpgrade.sol
6ca81e7796640015458083001b068fe2aa9cd56f	contracts/contract/dao/node/RocketDAONodeTrustedProposals.sol
ee9dcdc3cfc1375bd7c0a18e602b603118baaba	contracts/contract/dao/protocol/RocketDAOProtocolProposals.sol
9eb042d1c047dd258ab3785458e14e34b93bc82e	contracts/contract/dao/protocol/RocketDAOProtocol.sol
75e11937311100a134d055c6d683b286825b88b5	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsRewards.sol
76dd45aeebdade40cf3df12c3b6eaa4d60a31d44	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsInflation.sol

SHA-1 Hash	File
05ef6bb54e501a6bdd3718a387d92fd6e0ca570a	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsDeposit.sol
d8d177fbeb398efed9ae2cf46511e5614ab96872	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsNetwork.sol
bfa6f800c374736cf64fa842a6e5001d28635eb	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettings.sol
1cddd122b145bd100d5e25ce8486e69bb19d4a8	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsAuction.sol
820f43fea15eff788f5f85896cb52ee2d8810775	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsMinipool.sol
9145f779f88723787a5fe20a3228058e4fe9186a	contracts/contract/dao/protocol/settings/RocketDAOProtocolSettingsNode.sol
c692c30c4914517da9f87e0b2dccc7a9d4134b9b	contracts/contract/dao/protocol/RocketDAOProtocolActions.sol
c6b7566a67ed27afccb4e3b77ca1877167fe94d7	contracts/contract/dao/RocketDAOProposal.sol
597f7b6f0eb5b05cfb9b74f4af750e2afa07f11c	contracts/contract/helper/PenaltyTest.sol
bf503ce2a597cc582c255f39e9ecfb6a5362ffc2	contracts/contract/helper/RevertOnTransfer.sol
5692e5148551021f7a529190338e0276f2cc7343	contracts/contract/minipool/RocketMinipoolBondReducer.sol
872b8563fef2dbe77a3301a205ff3c39182cf1cb	contracts/contract/minipool/RocketMinipoolManager.sol
2fa1e7d7e69703ae18759bf4afa82a674420c01b	contracts/contract/minipool/RocketMinipoolQueue.sol
f393a16e1f674ade2ef9f1ecb86b930d18ae8d45	contracts/contract/minipool/RocketMinipoolStorageLayout.sol
cb683564f47dac2a13ba87622bf4f69d264f1cc6	contracts/contract/minipool/RocketMinipoolBase.sol
3df5f46799f97b7a383bc3cc8661adfe5aaa913a	contracts/contract/minipool/RocketMinipoolStatus.sol
08b9c9d65188d4931b24865c86795602a48247a2	contracts/contract/minipool/RocketMinipoolPenalty.sol
a12e8f5594c70b9841097836dc3954ddb97bf02e	contracts/contract/minipool/RocketMinipoolDelegate.sol
9fbadfca1a9d012bafbf6fff127fe38d589826d8	contracts/contract/minipool/RocketMinipoolFactory.sol
d4c47c746ef9fbdee7bef9cab11ccc8811478ed2	contracts/contract/auction/RocketAuctionManager.sol
fa66f01a46cc020e8195ebe8da572ee941a82df8	contracts/contract/util/AddressSetStorage.sol
538e2b9b6b13d1479226a2fe9cdd8dc2495b2514	contracts/contract/util/AddressQueueStorage.sol
5883ae7125d52f6f2257e0513300897f9a783343	contracts/contract/old/node/RocketNodeStakingOld.sol
e99e576cbdf38d8d6e3fbcdd5abd63362f5cc5c	contracts/contract/old/node/RocketNodeDistributorDelegateOld.sol
4163cf93214c71d46078ee8551863770069cc360	contracts/contract/old/node/RocketNodeManagerOld.sol
18be22f61a864272b7927d8015404a0a2cc7705c	contracts/contract/old/node/RocketNodeDepositOld.sol
c63733bf0da9f7a2750abc17c0399fd578c2f74e	contracts/contract/old/network/RocketNetworkPricesOld.sol
73249ffc350985908b54b498b5564d52613d7788	contracts/contract/old/dao/node/settings/RocketDAONodeTrustedSettingsMinipoolOld.sol
d52198de4d315e7e5f9277e3a6217540f098bb1b	contracts/contract/old/dao/protocol/settings/RocketDAOProtocolSettingsDepositOld.sol
22596ea8ae13053a6b71f25e0524c92694daf837	contracts/contract/old/dao/protocol/settings/RocketDAOProtocolSettingsNodeOld.sol
c824fb92cf1dd25326cf9f82048ed1ceca0d86df	contracts/contract/old/dao/protocol/settings/RocketDAOProtocolSettingsMinipoolOld.sol
731eed38ef605e4ab3f7e1b7f2885e3ebf5a55c8	contracts/contract/old/minipool/RocketMinipoolOld.sol
9be158bbfa5a787847468b92067f24d9eab1c6c	contracts/contract/old/minipool/RocketMinipoolManagerOld.sol
b56e74b5e1fd7692e71a2b33c77f7db6cb09bea3	contracts/contract/old/minipool/RocketMinipoolFactoryOld.sol
3b517c55a54b5234c8e5c5a5497b3ead4020dd8b	contracts/contract/old/minipool/RocketMinipoolQueueOld.sol
9ca01d01c6c8e3c41573f273c9c9545d3866aae4	contracts/contract/old/minipool/RocketMinipoolDelegateOld.sol
bdc6324139800c351bb70a0e7f47d5586c95a56e	contracts/contract/old/minipool/RocketMinipoolStorageLayoutOld.sol
200d56ccab76cc739055597e0af551f4a09add1c	contracts/contract/old/deposit/RocketDepositPoolOld.sol
78b524187aec0479e15178ded2d4f183c7414e02	contracts/contract/old/RocketNetworkFeesOld.sol
04cdfab0f47dd76a900ad25ef7a1995790007cbd	contracts/contract/deposit/RocketDepositPool.sol
6d556731d640f8ff78f04b3b2ba54ccc52b9791a	contracts/contract/token/RocketTokenRPL.sol
1fa2dd47d3d999602b866db678db083d700e0799	contracts/contract/token/temp/RocketTokenDummyRPL.sol
b9cea77bf1178201cc8e4b839016165d73740137	contracts/contract/token/RocketTokenRETH.sol
0e17a838a7ae3a0d117bdb29dedf77b16bce3736	contracts/contract/RocketStorage.sol
2b123f1a01b8fc7664336af46038d5db2f0117f3	contracts/types/MinipoolDetails.sol
5bd86e815d047d6aab8b091a0b783f1cd10de4e7	contracts/types/MinipoolDeposit.sol
31e943e327367421baf300e9b820823fafb94311	contracts/types/NodeDetails.sol
131fabe8d6efbed51f9c7f24ed88356c03315c9f	contracts/types/SettingType.sol
555564772a803a032929cb3d50091999170ec93c	contracts/types/RewardSubmission.sol
d57614fab9b875413a5afa2fa28837f005af067a	contracts/types/MinipoolStatus.sol
93d2dc04d17a72fcd9f5b8a736b62f524ee87106	contracts/types/old/NodeDetailsOld.sol
cfff9e6406d398e3fe838d001ffd91cdeffa0b88	contracts/interface/node/RocketNodeDistributorFactoryInterface.sol
dfde8dac4ab81303f3787b8ce9ea628773a8428d	contracts/interface/node/RocketNodeStakingInterface.sol

SHA-1 Hash	File
5c9cb11249756bfeac769deaf0c8533e5d381ca2	contracts/interface/node/RocketNodeManagerInterface.sol
c07bc1a65cc9a9beb06eb311d5345a7040bfb778	contracts/interface/node/RocketNodeDepositInterface.sol
2a993cfb99c99df89628c77e9edefefe55d1483a	contracts/interface/node/RocketNodeDistributorInterface.sol
f97a5ced28a49d32180441d7379457896f6bb825	contracts/interface/rewards/claims/RocketClaimDAOInterface.sol
808f2c9875eb438edf944ba564adc5ea31213b44	contracts/interface/rewards/claims/RocketClaimNodeInterface.sol
f9b774da2203be3833acb225b2ed1a57b06d4a46	contracts/interface/rewards/claims/RocketClaimTrustedNodeInterface.sol
8eb877d619690538c64b84c45c15077d3439501b	contracts/interface/rewards/RocketSmoothingPoolInterface.sol
5943539f2bae9ee3b4cd49cfd1bcb9e3c033c094	contracts/interface/rewards/RocketRewardsRelayInterface.sol
b2e6aabaccf88e3ac9e3a846e7df41ad64a691de	contracts/interface/rewards/RocketRewardsPoolInterface.sol
1cc38d548b5051fef88fb1fb8b2c5715cfd93048	contracts/interface/RocketStorageInterface.sol
f1c07b7eff6755965f2f6d8d04f5f8a160e73fbe	contracts/interface/network/RocketNetworkBalancesInterface.sol
66a93db7577f75799403201e116bc98b39753e66	contracts/interface/network/RocketNetworkFeesInterface.sol
23a03945ca61fde02619754dab79738ebc699cde	contracts/interface/network/RocketNetworkPricesInterface.sol
de391100daede5612fb913f904b64c84e5b5564a	contracts/interface/network/RocketNetworkPenaltiesInterface.sol
05b0dc75a1c8871ec3ff17c9a68f68335e344786	contracts/interface/dao/node/RocketDAONodeTrustedInterface.sol
1e2fcfeba5a1cb02520328d0cbb82640f51a5440	contracts/interface/dao/node/settings/RocketDAONodeTrustedSettingsInterface.sol
452c28b2b5db2b487013ae27a4dafd3826cc7aad	contracts/interface/dao/node/settings/RocketDAONodeTrustedSettingsMembersInterface.sol
8bd46a6e1f6988a332f6c1c6921ee566b3128db2	contracts/interface/dao/node/settings/RocketDAONodeTrustedSettingsRewardsInterface.sol
d74e01d9b27e2e544c498d80a43a4ca9eb559f75	contracts/interface/dao/node/settings/RocketDAONodeTrustedSettingsMinipoolInterface.sol
e51bda8eaf5d0adbed8ce748d50bf394194020a3	contracts/interface/dao/node/settings/RocketDAONodeTrustedSettingsProposalsInterface.sol
2b403337eb717c363a8c0d43f734e8bb2a52a428	contracts/interface/dao/node/RocketDAONodeTrustedProposalsInterface.sol
a060b82998383560e546e4f3d979648db36f8b35	contracts/interface/dao/node/RocketDAONodeTrustedActionsInterface.sol
12bf1b71506bcaa85c905796e1a45bf32cf4cd20	contracts/interface/dao/node/RocketDAONodeTrustedUpgradeInterface.sol
9611ef6727bd4660835d3d63a20b870d83091c57	contracts/interface/dao/RocketDAOProposalInterface.sol
954bdc0c88c5a2cc1f48e0d36094497573d8316c	contracts/interface/dao/protocol/RocketDAOProtocolProposalsInterface.sol
de87bf180b6fb7df2ccd6235e5d65d41f5adeed	contracts/interface/dao/protocol/RocketDAOProtocolActionsInterface.sol
1763f669eed40014cd6b9a5b6450ef8933d5b93d	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsNodeInterface.sol
71bb735546a78b0d5655362900099b6a48be09b2	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsDepositInterface.sol
239d0e9b16826d507e5969b576f0e9ae62c4c19b	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsNetworkInterface.sol
d7f3796a17e351037c5b26d5cce8813732f35478	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsInflationInterface.sol
f5a68583e26cbcecaae578e05a193b629a675971	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsAuctionInterface.sol
c02d578e41414d241ab10b082cec3fa72e61ea55	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsRewardsInterface.sol
d5fb57ee5e68482d1ef976d3f89c873d8b19c317	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsMinipoolInterface.sol
af57eb613c63ce3d47f812c96a42d024430961ac	contracts/interface/dao/protocol/settings/RocketDAOProtocolSettingsInterface.sol
c774f46eca674eff940a8879986495871486ce29	contracts/interface/dao/protocol/RocketDAOProtocolInterface.sol
9cced1a86b7063603337ea26351a5eab6172eeb4	contracts/interface/minipool/RocketMinipoolStatusInterface.sol
5a21b810b85ec99f4449580d1cb308f0829c3132	contracts/interface/minipool/RocketMinipoolInterface.sol
9612f16185357104a28835121f9c276ac41c3ac0	contracts/interface/minipool/RocketMinipoolBondReducerInterface.sol
c7454f8bc5474382369bd03313abb378852a855d	contracts/interface/minipool/RocketMinipoolManagerInterface.sol
a5ab18a608c884c09ed8646fa637b154ed475d7f	contracts/interface/minipool/RocketMinipoolPenaltyInterface.sol
8b7e3d06e9a221b769359bf43d4a8bf714700b21	contracts/interface/minipool/RocketMinipoolFactoryInterface.sol
54d241f1e4bf58715fc0e3c57e8689aa2fcfd03b	contracts/interface/minipool/RocketMinipoolQueueInterface.sol
b4384136cd0b9a0ada9fbbf4677e93508aa2102d	contracts/interface/RocketVaultInterface.sol
97be385ba8163a4d6e090162d265fc524ca4b4ed	contracts/interface/auction/RocketAuctionManagerInterface.sol
5cbb81c3088e73ddd70e14625df8692c0992ab7	contracts/interface/util/AddressQueueStorageInterface.sol
896dacb8f86aadcb1bcfe1d4fab102b10d2bc4fa	contracts/interface/util/AddressSetStorageInterface.sol
1a0dc382251ed0c61e264a618dda4690f9c1038c	contracts/interface/RocketVaultWithdrawerInterface.sol
e15b25e42ced34804a3bb9d8b703f49ac4abcd91	contracts/interface/old/RocketNodeDepositInterfaceOld.sol
a3b41c4e7b1a364cf06b8a3ef212d7711f8d5b7d	contracts/interface/old/RocketNodeManagerInterfaceOld.sol
393060945891e037d826c9eb68245ab032c8facb	contracts/interface/old/RocketDAOProtocolSettingsDepositInterfaceOld.sol
199b6247e297770a57f9194329d612c88450912	contracts/interface/old/RocketDAOProtocolSettingsMinipoolInterfaceOld.sol
443a05bb1d24b47f917fd82ccab11edf26275532	contracts/interface/old/RocketNodeStakingInterfaceOld.sol
fb07f75c8cc09480dca7d2d77f31f393a30768c	contracts/interface/old/RocketNetworkPricesInterfaceOld.sol

SHA-1 Hash	File
641fec69187057f8a6b61af1cd319d16691e31e7	contracts/interface/old/RocketMinipoolQueueInterfaceOld.sol
06dbe261b1e5845f8372dcf38960586671536f66	contracts/interface/old/RocketMinipoolManagerInterfaceOld.sol
8474c41be9da63a43932cf664549e80cd328936b	contracts/interface/old/RocketDAOProtocolSettingsNodeInterfaceOld.sol
c482301168243ce9fb2fb30437e4ef80c70150c0	contracts/interface/old/RocketMinipoolFactoryInterfaceOld.sol
c91cd7e324ebb5816c5777a5439d97ff5facfbfb	contracts/interface/old/RocketMinipoolInterfaceOld.sol
9b80affcc3e20f394a8e8d6a51e2b50f7c2e04a3	contracts/interface/old/RocketDAONodeTrustedSettingsMinipoolInterfaceOld.sol
0f076ba9160da4e99fce2f74d05de49baddf78d7	contracts/interface/old/RocketDepositPoolInterfaceOld.sol
b6869515a10a2ac8092bef1769a6f1e34b907715	contracts/interface/old/RocketNodeDistributorInterfaceOld.sol
ec56e4455efe3514697104258c32df13f3372469	contracts/interface/deposit/RocketDepositPoolInterface.sol
af4be5e3610b15b9670e1af4b3976b45885f85d4	contracts/interface/token/RocketTokenRPLInterface.sol
5527b3f945c2650b7efccbf44eb5f781f3a2e4f7	contracts/interface/token/RocketTokenRETHInterface.sol
2705faf4605281dfeee78c56847d09cb3100bf77	contracts/interface/casper/DepositInterface.sol

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.