Rocketpool v1.1 Features

Date	July 2022
Auditors	Dominik Muhs, Christian Goll, David Braun, Rai Yang

1 Executive Summary

This report presents the results of our engagement with **RocketPool** to review **their v1.1 changes**.

The review was conducted over two weeks, from **July 4** to **July 15** by **Dominik Muhs**, **Christian Goll**, **David Braun** and **Rai Yang**. A total of six person-weeks were spent.

During the first week, the audit team familiarized itself with all necessary components implementing the new in-scope features. The group investigated the penalty system, Minipool-related accounting, and the overall staking lifecycle.

In the second week, the auditors focused on exploring the new reward system and validating assumptions around the system's token economy. A small amount of time has furthermore been spent validating issues from the previous audit performed in March 2021. These issues are marked as "Acknowledged" since they were previously reviewed by the development team. These items have been added for completeness and visibility.

Given the time constraints, it was agreed to conduct this review on a best-effort basis prioritizing the focus areas.

2 Scope

Our review focused on the commit hash `f7657e64591597e45116c34245d2ef7eed8b8243`. Furthermore, the following Gist was provided: https://gist.github.com/kanewallmann/835d1f96c2754220382b85a637dec51e (revision from June 24)

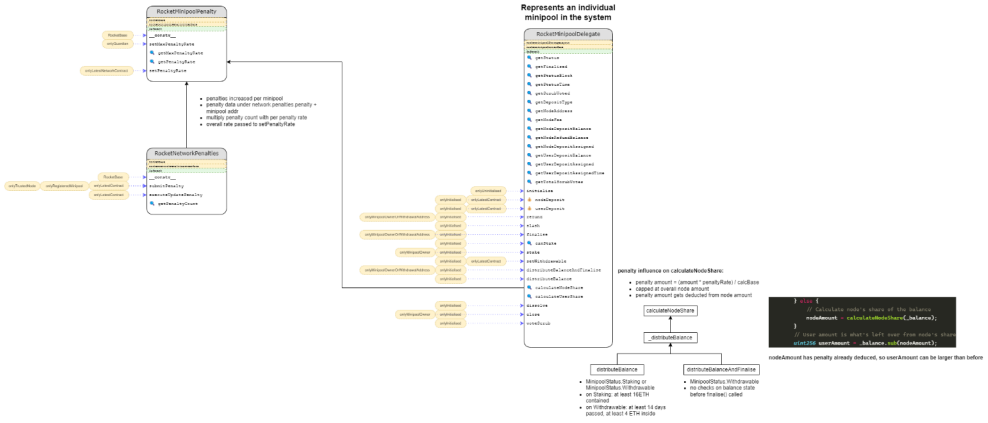
2.1 Objectives

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

1. The security of the fee distribution and penalty systems and the related oDAO functionalities.
2. The Merkle tree-based reward system and changes leading to its integration, e.g., in the node manager and node staking contracts.
3. The protocol's new deposit fee parameter and overall effort to prevent front-running attacks.
4. Ensure that the system is implemented consistently with the intended functionality and without unintended edge cases.
5. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 System Overview

The system components involved in applying and accounting for penalties.



4 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.

4.1 RocketMinipoolDelegate - Minipool owner's stake can get stuck Major

Description

Once a Minipool is initialized (`MinipoolStatus.Initialised`), the owner can no longer call the `dissolve` function to close the Minipool and retrieve their stake until the protocol assigns sufficient user funds. The user funds must be fully allocated, and the Minipool must proceed to the `MinipoolStatus.Prelaunch` state to successfully call `dissolve` . The Minipool owner's stake will be locked in the contract until then.

Examples

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L430-L435

```
require(
    (status == MinipoolStatus.Prelaunch && block.timestamp.sub(statusTime) >= rocketDAOProtocolSettingsMinipool.getLaunchTimeout()),
    "The minipool can only be dissolved once it has timed out"
);
// Perform the dissolution
_dissolve();
```

Recommendation

Adhere to the business logic outlined in the function's comment:

> Only accepts calls from the minipool owner (node), or from any address if timed out

Consequentially, the function should first check whether the `msg.sender` is the Minipool owner and, if not, fall back to the timeout check. The `Initialised` status must also be considered to allow owners an early exit before user funds have been assigned.

4.2 RocketStorage - Concentrated risk by allowing all registered contracts to change arbitrary settings Major Acknowledged

Description

The ACL for changing settings in the centralized `RocketStorage` allows any registered contract (listed under the `contract.exists` key) to change settings that belong to other parts of the system.

The concern is that if someone finds a way to add their malicious contract to the registered contact list, they will override any setting in the system. The storage is authoritative when checking specific ACLs. Setting any value might allow an attacker to gain control of the complete system. Allowing any contract to overwrite other contracts' settings dramatically increases the attack surface.

Minipool is self-destructed when closed, but it can be recreated by the Minipool owner referencing a malicious minipool delegate contract updated by a corrupted DAO vote. The Minipool owner can steal funds staked by the users.

Examples

code/contracts/contract/RocketStorage.sol:L41-L53

```
modifier onlyLatestRocketNetworkContract() {
    if (storageInit == true) {
        // Make sure the access is permitted to only contracts in our Dapp
        require(booleanStorage[keccak256(abi.encodePacked("contract.exists", msg.sender))], "Invalid or outdated network contract");
    } else {
        // Only Dapp and the guardian account are allowed access during initialisation.
        // tx.origin is only safe to use in this case for deployment since no external contracts are interacted with
        require((
            booleanStorage[keccak256(abi.encodePacked("contract.exists", msg.sender))] || tx.origin == guardian
        ), "Invalid or outdated network contract attempting access during deployment");
    }
    _;
}
```

code/contracts/contract/minipool/RocketMinipoolDelegate.sol:L466

```
selfdestruct(payable(rocketTokenRETH));
```

code/contracts/contract/minipool/RocketMinipool.sol:L37

```
address delegateAddress = getContractAddress("rocketMinipoolDelegate");
```

Recommendation

Allow contracts to only change settings related to their namespace.

4.3 RocketMerkleDistributorMainnet - Lacking input validation in `claimAndStake` and `_claim`

Medium

Description

In the `RocketMerkleDistributorMainnet` contract, the `claimAndStake` function allows node operators to claim rewards for one or more reward intervals, and stake a set amount of RPL at the same time.

The internal `_claim` function does not check whether the length of its input arrays (`_rewardIndex` , `_amountRPL` , `_amountETH` , `_merkleProof`) are all equal. Instead, it loops over the `_rewardIndex` array right away, using the current index to access other parameters' entries. If any array other than `_rewardIndex` contains more fields, they will not be considered. If the `_rewardIndex` array is too short, the transaction will revert due to an out-of-bounds error.

Examples

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L84-L87

```
for (uint256 i = 0; i < _rewardIndex.length; i++) {
    totalAmountRPL = totalAmountRPL.add(_amountRPL[i]);
    totalAmountETH = totalAmountETH.add(_amountETH[i]);
}
```

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L122-L143

```
for (uint256 i = 0; i < _rewardIndex.length; i++) {
    // Prevent accidental claim of 0
    require(_amountRPL[i] > 0 || _amountETH[i] > 0, "Invalid amount");
    // Check if this entry has a different word index than the previous
    if (indexWordIndex != _rewardIndex[i] / 256) {
        // Store the previous word
        setUint(claimedWordKey, claimedWord);
        // Load the word for this entry
        indexWordIndex = _rewardIndex[i] / 256;
        claimedWordKey = keccak256(abi.encodePacked('rewards.interval.claimed', _nodeAddress, indexWordIndex));
        claimedWord = getUint(claimedWordKey);
    }
    // Calculate the bit index for this entry
    uint256 indexBitIndex = _rewardIndex[i] % 256;
    // Ensure the bit is not yet set on this word
    uint256 mask = (1 << indexBitIndex);
    require(claimedWord & mask != mask, "Already claimed");
    // Verify the merkle proof
    require(_verifyProof(_rewardIndex[i], _nodeAddress, _amountRPL[i], _amountETH[i], _merkleProof[i]), "Invalid proof");
    // Set the bit for the current reward index
    claimedWord = claimedWord | (1 << indexBitIndex);
}
```

Recommendation

Check the length of input arrays (`_rewardIndex` , `_amountRPL` , `_amountETH` , `_merkleProof`) to be of equal length and revert with a meaningful error message informing the executing party that the submitted parameters are malformed.

4.4 RocketRewardsPool - Incorrect modifier Minor

Description

The `RocketRewardsPool.executeRewardSnapshot` function has the following `onlyLatestContract` modifier:

code/contracts/contract/rewards/RocketRewardsPool.sol:L185-L186

```
// Executes reward snapshot if consensus threshold is reached
function executeRewardSnapshot(RewardSubmission calldata _submission) override external onlyLatestContract("rocketNetworkBalances", ad
```

It was probably copied and pasted by mistake from a method of the `rocketNetworkBalances` contract. As written, it will prevent `executeRewardSnapshot` from executing because it is checking for the wrong contract (`rocketNetworkBalances`).

Recommendation

Rewrite the modifier to check for `rocketRewardsPool` :

```
onlyLatestContract("rocketRewardsPool", address(this))
```

A failed unit test could have caught this bug. We recommend adopting Test-Driven Development to increase test coverage.

4.5 Redundant interface casts Minor

Description

Throughout the code, various redundant and duplicate interface casts exist. Furthermore, these occurrences often violate the project's naming conventions, where contract instances have variable names ending with `Contract` and address types carrying the `Address` suffix.

Examples

code/contracts/contract/dao/node/RocketDAONodeTrustedActions.sol:L105-L106

```
// Let vault know it can move these tokens to itself now and credit the balance to this contract
rocketVault.depositToken(getContractName(address(this)), IERC20(rocketTokenRPLAddress), rplBondAmount);
```

code/contracts/contract/minipool/RocketMinipoolManager.sol:L205-L207

```
function getMinipoolWithdrawalCredentials(address _minipoolAddress) override public pure returns (bytes memory) {
    return abi.encodePacked(byte(0x01), bytes11(0x0), address(_minipoolAddress));
}
```

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L92-L94

```
if (remaining > 0) {
    rocketVault.withdrawToken(withdrawalAddress, IERC20(rocketTokenRPLAddress), remaining);
}
```

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L105-L107

```
RocketNodeStakingInterface rocketNodeStaking = RocketNodeStakingInterface(getContractAddress("rocketNodeStaking"));
rocketVault.withdrawToken(address(this), IERC20(rocketTokenRPLAddress), _stakeAmount);
rocketTokenRPL.approve(address(rocketNodeStaking), _stakeAmount);
```

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);
```

code/contracts/contract/token/RocketTokenRPL.sol:L183-L189

```
IERC20 rplInflationContract = IERC20(address(this));
// Get the current allowance for Rocket Vault
uint256 vaultAllowance = rplFixedSupplyContract.allowance(rocketVaultAddress, address(this));
// Now allow Rocket Vault to move those tokens, we also need to account of any other allowances for this token from other contracts in
require(rplInflationContract.approve(rocketVaultAddress, vaultAllowance.add(newTokens)), "Allowance for Rocket Vault could not be appr
// Let vault know it can move these tokens to itself now and credit the balance to the RPL rewards pool contract
rocketVaultContract.depositToken("rocketRewardsPool", IERC20(address(this)), newTokens);
```

In the above, furthermore, `rplInflationContract` can be removed and the call to `rplInflationContract.approve` refactored to `approve` since the main contract inherits from `ERC20Burnable`.

Recommendation

Remove redundant casts, work with the most specific types where possible, and remove any state variables that become unused in the process.

4.6 RocketVault - Confusing parameter type handling Minor

Description

In the `RocketVault.withdrawToken` function, an `IERC20` parameter `_tokenAddress` is passed. The ERC20 interface type is then used to determine `contractKey`, an internal key for ERC20 token balance tracking.

Right after, the interface instance is explicitly cast into the same type again. This cast indicates a potential error made during a previous refactoring. Either the cast is redundant since the `contractKey` works with the interface instance, or `_tokenAddress` is supposed to be an `address` type.

Examples

code/contracts/contract/RocketVault.sol:L98-L111

```
function withdrawToken(address _withdrawalAddress, IERC20 _tokenAddress, uint256 _amount) override external onlyLatestNetworkContract
    // Valid amount?
    require(_amount > 0, "No valid amount of tokens given to withdraw");
    // 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
    IERC20 tokenContract = IERC20(_tokenAddress);
    // Withdraw to the desired address
    require(tokenContract.transfer(_withdrawalAddress, _amount), "Rocket Vault token withdrawal unsuccessful");
    // Emit token withdrawn event
    emit TokenWithdrawn(contractKey, address(_tokenAddress), _amount, block.timestamp);
}
```

Recommendation

Change the `_tokenAddress` parameter to `_tokenContract` and consistently work with the specific ERC20 interface. When an address type is needed, the explicit `address(_tokenContract)` cast can be used.

4.7 RocketDepositPool - risk of gas-based denial of service Minor Acknowledged

Description

`RocketDepositPool._assignDeposits` seems to be a gas-heavy function, with many external calls, many of which are inside the main for-loop. By default, `_rocketDAOProtocolSettingsDeposit.getMaximumDepositAssignments();` returns `2`, which is not a security concern. Through a DAO vote, however, the `deposit.assign.maximum` settings key can be set to a value that exhausts the block gas limit and effectively deactivates the deposit assignment process.

Examples

code/contracts/contract/deposit/RocketDepositPool.sol:L155-L160

```
uint256 maxAssignments = _rocketDAOProtocolSettingsDeposit.getMaximumDepositAssignments();
MinipoolAssignment[] memory assignments = new MinipoolAssignment[](maxAssignments);
MinipoolDeposit depositType = MinipoolDeposit.None;
uint256 count = 0;
uint256 minipoolCapacity = 0;
for (uint256 i = 0; i < maxAssignments; ++i) {
```

Recommendation

A check that prevents a DAO vote from setting unreasonably high `deposit.assign.maximum` values should be added.

4.8 RocketMerkleDistributorMainnet - Improve `_claim` tests

Description

The `_claim` function does some fairly complex gas-optimization bitmapping to record which rewards have been claimed, but the test coverage of this code is thin and doesn't, e.g., test any of the three `require` cases.

Examples

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L123-L124

```
// Prevent accidental claim of 0
require(_amountRPL[i] > 0 || _amountETH[i] > 0, "Invalid amount");
```

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L136-L138

```
// Ensure the bit is not yet set on this word
uint256 mask = (1 << indexBitIndex);
require(claimedWord & mask != mask, "Already claimed");
```

code/contracts/contract/rewards/RocketMerkleDistributorMainnet.sol:L139-L140

```
// Verify the merkle proof
require(_verifyProof(_rewardIndex[i], _nodeAddress, _amountRPL[i], _amountETH[i], _merkleProof[i]), "Invalid proof");
```

Recommendation

Improve the test coverage of the complex `_claim` function in `test/rewards/rewards-test.js` to ensure the bitmap math is correct and that the `require` cases fail when appropriate.

4.9 Gas Optimizations

Description

The RocketPool developers expressed the need for gas-efficient code in their lite specification of the contract's changes here; thus, we recommend the following gas optimizations for the for loops used throughout the codebase, particularly in `RocketRewardsPool`.

Examples

code/contracts/contract/rewards/RocketRewardsPool.sol:L110-L119

```
function getClaimingContractsPerc(string[] memory _claimingContracts) override external view returns (uint256[] memory) {
    // Load contract
    RocketDAOProtocolSettingsRewardsInterface daoSettingsRewards = RocketDAOProtocolSettingsRewardsInterface(getContractAddress("rocke
    // Get the % amount allocated to this claim contract
    uint256[] memory percentages = new uint256[](_claimingContracts.length);
    for (uint256 i = 0; i < _claimingContracts.length; i++){
        percentages[i] = daoSettingsRewards.getRewardsClaimerPerc(_claimingContracts[i]);
    }
    return percentages;
}
```

Recommendation

In the above example, it is far more efficient to cache the array length accessed and use a pre-increment over a post-increment. The resulting modified code will look like the one below. Comments have been attached indicating the modifications.

```
function getClaimingContractsPerc(string[] memory _claimingContracts) override external view returns (uint256[] memory) {
     // Load contract
     RocketDAOProtocolSettingsRewardsInterface daoSettingsRewards = RocketDAOProtocolSettingsRewardsInterface(getContractAddress("rock
     // Get the % amount allocated to this claim contract
     uint256[] memory percentages = new uint256[](_claimingContracts.length);
     // Cache the array length
     uint256 arrayLength = _claimingContracts.length;
     // Utilise post increment
     for (uint256 i = 0; i < _claimingContracts.length; ++i){
         percentages[i] = daoSettingsRewards.getRewardsClaimerPerc(_claimingContracts[i]);
     }
     return percentages;
 }
```

This recommendation is based on Harikrishnan Mulackal's gas optimization write-up. Please refer to his document for further details.

4.10 RocketMinipoolQueue - Confusing function naming

Description

In the `RocketMinipoolQueue` code, the `getLength` function has been overloaded to use a `MinipoolDeposit` parameter on the one hand and a `bytes32` parameter on the other hand. The function names should be explicitly distinguished to increase maintainability and readability.

Examples

code/contracts/contract/minipool/RocketMinipoolQueue.sol:L49

```
function getLength(MinipoolDeposit _depositType) override external view returns (uint256) {
```

code/contracts/contract/minipool/RocketMinipoolQueue.sol:L55

```
function getLength(bytes32 _key) private view returns (uint256) {
```

Recommendation

We recommend making the function names more expressive to help readers distinguish between the respective business logic. Line 49 could be `getLengthForDepositType`, and line 55 could be `getLengthForKey` or, following the code base's existing pattern, adding an underscore to underline the function's visibility: `_getLength`.

4.11 Potential for collisions when writing/reading settings

Description

The system heavily relies on a centralized hash indexed storage. The storage keys are formed with distinct namespace prefixes potentially concatenated with user-tainted input. Here's one example of this:

code/contracts/contract/dao/RocketDAOProposal.sol:L173-L186

```
setAddress(keccak256(abi.encodePacked(daoProposalNameSpace, "proposer", proposalID)), _member);                    // Which member is
setString(keccak256(abi.encodePacked(daoProposalNameSpace, "dao", proposalID)), _dao);                             // The DAO the pro
setString(keccak256(abi.encodePacked(daoProposalNameSpace, "message", proposalID)), _message);                     // A general messa
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "start", proposalID)), _startTime);                       // The time the pr
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "end", proposalID)), endTime);                            // The time the pr
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "expires", proposalID)), expires);                        // The time when t
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "created", proposalID)), block.timestamp);                // The time the pr
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "votes.for", proposalID)), 0);                            // Votes for this
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "votes.against", proposalID)), 0);                        // Votes against t
setUint(keccak256(abi.encodePacked(daoProposalNameSpace, "votes.required", proposalID)), _votesRequired);          // How many votes
setBool(keccak256(abi.encodePacked(daoProposalNameSpace, "cancelled", proposalID)), false);                        // The proposer ca
setBool(keccak256(abi.encodePacked(daoProposalNameSpace, "executed", proposalID)), false);                         // Has this propos
setBytes(keccak256(abi.encodePacked(daoProposalNameSpace, "payload", proposalID)), _payload);                      // A calldata payl
// Update the total proposals
```

`abi.encodePacked` encodes dynamic types in-place without a length prefix, and static types will not be padded if they are shorter than 32 bytes. If namespace prefixes are not chosen carefully, a user might provide a value that overlaps into another settings namespace (after the prefix). Special care should be taken if dynamic or short types are used with `encodePacked` as this might make it easier to force such situations.

Throughout the review, the assessment team has not found any signs of this issue. However, it should be noted that developers must be made aware of this potential problem. It is highly recommended to support the secure development process by tooling that checks for possible overlaps in the CI pipeline.

4.12 Consistent documentation using NatSpec

Description

For consistency and user- and machine-readability, we recommend using the NatSpec format: https://docs.soliditylang.org/en/v0.7.6/natspec-format.html.

4.13 RocketTokenRETH - Evasion of `receive` event emission

Description

When the rETH token contract receives ETH funds, it is expected to emit an `EtherDeposited` event. The same event is raised in the `depositExcess` function. The event might be consumed by off-chain infrastructure for accounting purposes. To disrupt that process, a malicious party can force-feed Ether into the rETH token contract through a large variety of means, increasing its ETH balance but not triggering an event.

Examples

code/contracts/contract/token/RocketTokenRETH.sol:L33-L36

```
receive() external payable {
    // Emit ether deposited event
    emit EtherDeposited(msg.sender, msg.value, block.timestamp);
}
```

Recommendation

We recommend double-checking off-chain infrastructure consuming the `EtherDeposited` event for potential force-feeding issues and taking the address balance into account instead of solely relying on the emitted event.

4.14 Create architecture diagram

Description

The Rocket Pool is composed of 40 smart contracts that collectively comprise over 5,000 lines of code. A clear understanding of the large code base is vital to discern any existing vulnerabilities and avoid introducing new ones.

Recommendation

Create an architecture diagram of the code to show the high-level structure and relationships between major components. This map will aid developers, especially ones unfamiliar with the code (such as external auditors), to come up to speed more quickly in understanding the code and serve as a collaboration tool for communication with other developers.

Appendix 1 - 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.