yAudit Incubator DAO Review

Review Resources:

- Governance proposal with background information
- ROOK governance process, inherited by Incubator DAO

Auditors:

- engn33r
- spalen

Table of Contents

Review Summary

Incubator DAO

Incubator DAO provides a way for ROOK token holders to redeem their ROOK for a fraction of the former ROOK DAO's treasury balance. This process is outlined in this governance proposal.

The contracts of Incubator DAO Repo were reviewed over 3 days. The code review was performed by 2 auditors between April 10 and April 12, 2023. The repository was nearly ready for deployment at the time of the review and the review was limited to the latest commit at the start of the review. This was commit 80091e9be3f8ce1a4cd36ff407f20952f1f000c5 for the Incubator DAO repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- exchangeROOK.sol
- exchangepROOK.sol
- prook.sol

There were no tests provided with the repository, but the auditors wrote tests that additionally performed fuzzing to confirm the proper functionality of all contracts. After the findings were presented to the Incubator DAO team, fixes were made and included in the code.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Incubator DAO and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	The contracts are not upgradeable, but the multisig has considerable control over the treasury assets and redeemed ROOK and pROOK held in the exchange contracts. However, this is part of the design of the contracts.
Mathematics	Good	There is no complex math involved.

Category	Mark	Description
Complexity	Good	The contract have very low complexity due to the low number of lines of code.
Libraries	Good	Only common OpenZeppelin libraries are used, which reduces the complexity of the code.
Decentralization	Low	The multisig holds a privileged position in all contracts which means the power is centralized in the 4 multisig signers. Value manipulation could occur by the multisig if they act maliciously.
Code stability	Good	The code is nearly ready for mainnet and does not need many adjustments.
Documentation	Good	The code had NatSpec comments on all functions to explain the contract logic.
Monitoring	Average	Events were added to most but not all important state change functions.
Testing and verification	Low	There were no tests included in the repository, but the auditors wrote up and provided tests.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - On-chain contracts assume off-chain trust

The governance proposal outlining the logic that Incubator DAO will follow indicates several parties that are excluded from redeeming ROOK in the Incubator DAO contracts. There is no on-chain logic preventing these parties from interacting with the Incubator DAO contracts, so the Incubator DAO contract design is assuming a level of off-chain trust between the involved parties.

There is also the reverse scenario where the Incubator DAO multisig can withdraw ROOK from exchangeROOK.sol and redeem it at the fixed exchange rate until the USDC is drained from exchangeROOK.sol. In effect the Incubator DAO multisig must be trusted to be a non-malicious party.

Technical Details

The 90-day phase 1 period of the Incubator DAO ROOK rage quit process will involve swapping ROOK for pROOK at a 1-to-1 exchange rate. Along with the pROOK token, USDC will also be received at a rate specified by the `exchangeRate` value. The ROOK token will effectively be pegged to a specific number of USDC tokens during Phase 1. There are some ROOK holders that are excluded from participating in this redemption process according to the governance proposal, but there is no logic in the exchangeROOK.sol contract to prevent these parties from interacting with the contract. This could result in the draining of USDC held in the contract by these excluded parties, which would effectively cause a bank run on the exchangeROOK.sol contract because some ROOK holders will not be able to receive their promised USDC at the set exchange rate. However, it should be noted that the evidence of these acts would be visible on-chain and there is a level of established trust between the ROOK holders which allowed the current governance proposal to be agreed on.

Another scenario that would have a negative impact for ROOK holders would involve the Incubator DAO multisig withdrawing ROOK from exchangeROOK.sol using

`withdrawAssets()` to take all the USDC from this contract before other ROOK holders can redeem their tokens. By depositing ROOK into exchangeROOK.sol to get USDC, then using `withdrawAssets()` to withdraw the ROOK, then depositing ROOK into exchangeROOK.sol ad infinitum, the USDC would get drained from exchangeROOK.sol. This would also effectively cause a bank run on the exchangeROOK.sol contract because some ROOK holders will not be able to receive their promised USDC at the set exchange rate.

An alternative approach for the Incubator DAO multisig to gain value is to withdraw ROOK from exchangeROOK.sol using `withdrawAssets()`, then to swap the ROOK tokens in a liquidity pool before the ROOK token has no value. While the liquidity pool would be imbalanced and be arbitraged due to the set `exchangeRate` value in exchangeROOK.sol, the multisig could once again take any new ROOK tokens deposited into exchangeROOK.sol and swap it into a liquidity pool again until the exchangeROOK.sol holds no more USDC.

Impact

High. While there is the possibility of value removal from the contract from excluded parties, such acts would break the rules of the rage quit agreement process and be clearly visible on-chain. These disincentives and the trust established between ROOK token holders provide some level of disincentive against bad actors.

Recommendation

Change the design of Phase 1 to only allow ROOK holders to redeem ROOK for pROOK. This suggestion would remove the ability to receive USDC in Phase 1 and move this to Phase 2. Phase 2 would start only after the 90-day Phase 1 period ends and allow pROOK holders to redeem their pROOK for USDC. This would prevent a direct ROOK => USDC exchange in Phase 1 and provide a time gap to confirm no bad actors exchanged their ROOK against the rules of the governance agreement. To show this proposed logic visually:

Phase 1: ROOK => pROOK Phase 2: pROOK => USDC

This suggested solution would still have problems if excluded ROOK is exchanged for pROOK during the 90-day Phase 1 period, because there may be a need for a restart of the 90-day Phase 1 period. But at least the bad actors would be identified and stopped before they could extract any value from the USDC treasury holdings.

Developer Response

Acknowledged, won't change. While this is a high operational risk rather than a technical risk, all parties involved in the governance decision (Rook Labs and ROOK governance token holders) have sufficient shared trust to continue forward with the current approach. Additional protective measures would add a delay in ROOK holders receiving the assets they are due.

Medium Findings

None.

Low Findings

1. Low - Contract owner can manipulate value received by ROOK or pROOK holders

The contract owner for the Incubator DAO contract holds a privileged role. This role and the functions that have the `onlyOwner` modifier could allow for value manipulation.

Technical Details

The contract owner will be a 3-of-4 multisig and should generally be a trusted party. However, if this assumption does not hold true, there are several ways the owner could manipulate the value received by ROOK or pROOK token holders:

1. `withdrawAssets()` could take out all the USDC from the exchangeROOK.sol or exchangepROOK.sol contracts.
2. The `mint()` function in prook.sol can be called at any time for any amount. The owner could mint many pROOK tokens at the start of Phase 2 and sweep all the USDC from this contract before other token holders get the chance to redeem their pROOK.
3. The `setExchangeRate()` function could be called to change the ROOK-USDC or pROOK-USDC `exchangeRate` value in exchangeROOK.sol or exchangepROOK.sol, giving some users more or less USDC in return for their ROOK tokens.

Impact

Low. The contract owner should be a trusted party, but if the owner acts maliciously, there are several ways in which ROOK or pROOK token holders can receive suboptimal exchanges of their tokens.

Recommendation

Consider limiting the privileges of the contract owner with timelocks for certain actions or allowing certain functions to be called only until a certain date. The latter suggestion could be implemented with a line of code such as `require(block.timestamp < 1683000000)`. More specifically, the minting of new pROOK tokens could be prevented after the 90 day Phase 1 period outlined in the proposal by only allowing `pROOK.mint()` to be called by the contract owner before a certain timestamp. Additionally, the `setExchangeRate()` functions could be removed from the exchange contracts because the `exchangeRate` can be set in the contract constructor.

Developer Response

Acknowledged, no change. The contract owner is a trusted party with high security standards (3-of-4 multisig). There is no plan to mint new pROOK tokens after the first mint, but given the high degree of off-chain trust required in this process, keeping flexibility could prove valuable. Furthermore, `exchangeRate` could still be modified by pausing the current exchange smart contract and deploying a new one with a different exchange rate.

2. Low - Remove ERC20Votes.sol import

The voting for Incubator DAO will be done on snapshot.org using the erc20-balance-of voting strategy, so there is no need to include code for on-chain voting like what is found in ERC20Votes.sol if it will not be used.

Technical Details

Incubator DAO will use the incubdao.eth snapshot.org space for governance voting. Because the governance approach will follow the approach used by ROOK, the same erc20-balance-of snapshot.org voting strategy will be used. This means that there is no need for the pROOK token to include voting logic on-chain, because the token balance held by an address will determine the votes that address receives.

Impact

Low. Adding code adds complexity. If the extra complexity does not provide any benefit, it should be removed to reduce potential security flaws in the additional code.

Recommendation

Remove the ERC20Votes.sol import to simplify the pROOK token and save gas on deployment.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

3. Low - Limit pROOK burn functions

The pROOK token inherits OpenZeppelin's ERC20Burnable.sol logic. This contract allows users to burn their pROOK tokens. It may be better to prevent users from burning their own pROOK tokens and instead only allow the owner of the pROOK contract to burn tokens.

Technical Details

If a user burns their pROOK token, they will lose the chance to redeem their pROOK for USDC during Phase 2 of the Incubator DAO rage quit process. Because ERC20Burnable.sol includes a public `burnFrom()` function, which only requires an allowance to be given to burn pROOK tokens, it's possible that scam websites could attempt to phish users to sign a pROOK allowance and burn their tokens during Phase 1 so that pROOK holders in Phase 2 would be allocated more USDC. This is because the `exchangeRate` in Phase 2 will be set to `remaining_USDC / pRook.totalSupply()`, and burning pROOK would reduce the `totalSupply` value.

Impact

Low. There is no use case for pROOK holders to burn their pROOK, only the multisig needs to burn pROOK after Phase 1 ends. To avoid any user error on the part of pROOK holders, remove the ability of pROOK holders to burn their pROOK except for the pROOK contract owner, which is the multisig.

Recommendation

The easiest approach would be to remove the import of `ERC20Burnable.sol` in prook.sol and add the following function protected by `onlyOwner` to prook.sol (the same approach used for `mint()`):

```
function burn(uint256 amount) public onlyOwner {
            _burn(_msgSender(), amount);
}
```

This function would allow only the multisig to burn pROOK tokens that they hold, which is done after Phase 1 when the multisig withdraws pROOK from exchangeROOK.sol. This is really the only use case where pROOK should be burned, so there is no need to give all

pROOK holders the ability to burn these tokens.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

Gas Savings Findings

1. Gas - Remove SafeMath import

The contracts are using solidity version 0.8.X, which includes overflow and underflow protection, making the SafeMath library import unnecessary.

Technical Details

Solidity 0.8.0 introduced a breaking change to implement overflow and underflow protection.

This means the SafeMath imports can be removed to save gas on deployment:

1 exchangeROOK.sol import
2 exchangepROOK.sol import

Impact

Gas savings.

Recommendation

Remove SafeMath imports from contracts. Modify the code, particularly the `exchange()` functions, to use standard arithmetic operators like `*` and `/` instead of `.mul()` and `.div()`.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

2. Gas - Declare variables immutable when possible

Using immutable variables can provide gas savings compared to non-immutable variables if the variables only need to be set once.

Technical Details

The token variables in exchangeROOK.sol and exchangepROOK.sol can be immutable. They are only set in the constructor and are not changed after that.

1. exchangeROOK.sol token variables
2. exchangepROOK.sol token variables

Impact

Gas savings.

Recommendation

Declare token variables immutable for gas savings.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

3. Gas - Remove ERC20.sol import from exchange contracts

exchangeROOK.sol and exchangepROOK.sol do not need to import the OpenZeppelin ERC20.sol contract. Instead, IERC20.sol can be used, which is already imported by SafeERC20.sol.

Technical Details

The ERC20.sol import in the exchange contracts is not needed and will spend unnecessary gas on deployment, so it can be removed.

1. exchangeROOK.sol ERC20 import
2. exchangepROOK.sol ERC20 import

Impact

Gas savings.

Recommendation

Remove the ERC20 import in the exchange contracts and make the following change when using SafeERC20:

```
- using SafeERC20 for ERC20;
+ using SafeERC20 for IERC20;
```

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

Informational Findings

1. Informational - Use consistent solidity versions

The solidity files in the repository are using different solidity versions. Ideally the contracts should use consistent solidity versions.

Technical Details

exchangeROOK.sol and exchangepROOK.sol use solidity 0.8.12, but prook.sol uses solidity ^0.8.9. Because different versions of solidity contain minor differences, and because the newer versions of solidity contain bug fixes and gas optimizations, it is generally recommended to use one of the most recent solidity versions.

Impact

Informational.

Recommendation

Use a consistent solidity version in all contracts, for example solidity 0.8.19.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

2. Informational - Add event to `setExchangeRate()`

`setExchangeRate()` modifies the value of `exchangeRate` which is a state variable. It can be helpful to add events for any action that modifies state variables to make it easier to trace when the value change happened and to add monitoring of such changes more easily.

Technical Details

Consider adding a `ExchangeRateSet` event in:

```
1 exchangeRook.setExchangeRate()
2 exchangepRook.setExchangeRate()
```

Impact

Informational.

Recommendation

Add an event to the `setExchangeRate()` functions in exchangeRook.sol and exchangepRook.sol.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

3. Informational - Make public functions external

A public function can be called internally and externally. An external function can only be called externally. If a public function does not need to be called internally, make it an external function. This can also save gas on deployment in some solidity versions.

Technical Details

Declare these public functions as external because they do not need to be called internally:

1. `mint()` in prook.sol
2. `pause()` in exchangeROOK.sol and in exchangepROOK.sol
3. `unpause()` in exchangepROOK.sol and in exchangepROOK.sol

Impact

Informational.

Recommendation

Declare public functions as external when possible.

Developer Response

@ibuygovernancetokens: Fixed in commit f88929b.

4. Informational - Incorrect values in governance proposal

The numbers in the equation put forth by the governance proposal do no match the current on-chain values.

Technical Details

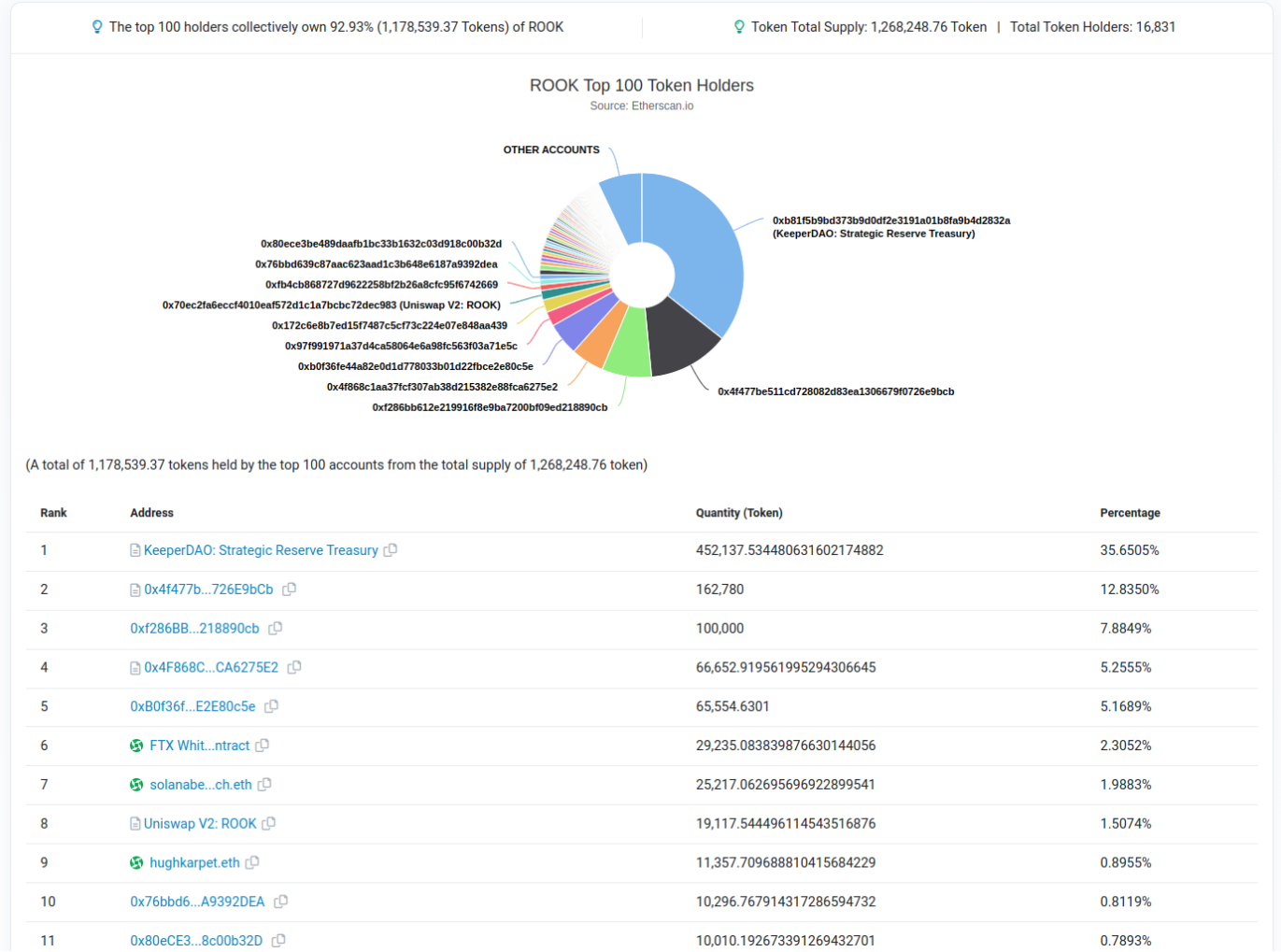
The governance proposal states that the exchangeROOK.sol exchangeRate will be set by the following equation:

> an exchange rate defined by Incubator DAO total treasury ÷ (ROOK total supply [1,268,253.79] - DAO owned ROOK [162,807.40] - Strategic Reserve owned ROOK [452,137.53] - DAO contributors owned ROOK [35,175.14])

The first number in this equation is inaccurate and is off by about 5 ROOK. The current on-

chain totalSupply value of ROOK token is 1268248.759, which is visible with etherscan or with: `cast --from-wei $(cast call 0xfa5047c9c78b8877af97bdcb85db743fd7313d4a "totalSupply()(uint256)")`

The second number in this equation is also inaccurate. Based on the top ROOK holders etherscan chart, the 2nd largest ROOK holder, which we will assume is the DAO owned ROOK tokens, holds 162,780 ROOK. This is a difference of roughly 27 ROOK from the value in the governance proposal.



Rank	Address	Quantity (Token)	Percentage
1	KeeperDAO: Strategic Reserve Treasury	452,137.534480631602174882	35.6505%
2	0x4f477b...726E9bCb	162,780	12.8350%
3	0xf286BB...218890cb	100,000	7.8849%
4	0x4F868C...CA6275E2	66,652.919561995294306645	5.2555%
5	0xB0f36f...E2E80c5e	65,554.6301	5.1689%
6	FTX Whit...ntract	29,235.083839876630144056	2.3052%
7	solanabe...ch.eth	25,217.062695696922899541	1.9883%
8	Uniswap V2: ROOK	19,117.544496114543516876	1.5074%
9	hughkarpet.eth	11,357.709688810415684229	0.8955%
10	0x76bbd6...A9392DEA	10,296.767914317286594732	0.8119%
11	0x80eCE3...8c00b32D	10,010.192673391269432701	0.7893%

The governance proposal does use the correct value for the Strategic Reserve owner ROOK of 452,137.53. The DAO contributors owned ROOK value is not easily checked at the time of the audit because the tokens are held by different addresses but will be combined into a multisig.

Impact

Informational.

Recommendation

Choose between using the latest on-chain values in the `exchangeRate` calculation when the exchangeROOK.sol contract is deployed, or trusting the numbers in the governance proposal.

Developer Response

Acknowledged, will use the latest on-chain values in the `exchangeRate` calculation.

Final remarks

The contracts are concise and clear. The logic aligns with the intended goals outlined in the governance proposal. The pROOK token is a simple ERC20 token with logic that matches the outputs of the OpenZeppelin contract wizard. The exchangeROOK.sol and exchangepROOK.sol contracts have `withdrawAssets()` and `setExchangeRate()` functions that provide a multisig-controlled escape hatch if there are problems with the outlined ROOK token redemption process. Although there is one high risk finding, it was previously acknowledged and discussed by the involved parties during the governance discussions. The other suggested edits are minor but could provide extra assurance to ROOK token holders involved in this process.