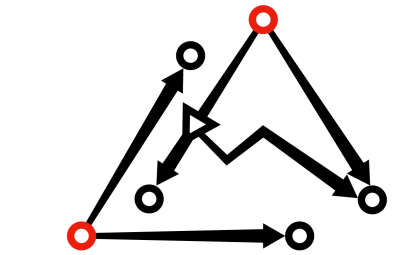

Least Authority

PRIVACY MATTERS

Smart Contracts

Security Audit Report

NEOKingdom DAO

Updated Final Audit Report: 25 September 2023

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: Deposited Tokens Can Be Redeemed
    - Issue B: Unsettled Deposits Can Be Locked
    - Issue C: Missing Modifier Preventing the Update of Non-Existent Resolutions
    - Issue D: The Status of InternalMarket or ShareholderRegistry Can Be Set to Contributor Status
    - Issue E: settleTokens Function Mints Extra NEOK Tokens to the GovernanceToken Smart Contract (Known Issue)
  - Suggestions
    - Suggestion 1: Improve Code Comments and Update the Documentation
    - Suggestion 2: Add Check To Verify Token Transfer Return Value
    - Suggestion 3: Add Zero Address Checks
    - Suggestion 4: Remove Redundant Checks
    - Suggestion 5: Use an Updated and Non-Floating Pragma Version Consistently Across the Project
    - Suggestion 6: Implement the Appropriate Interface
    - Suggestion 7: Use Custom Error To Output Arguments in Error Messages
- About Least Authority
- Our Methodology

Overview

Background

NEOKingdom DAO has requested that Least Authority perform a security audit of their smart contacts.

Project Dates

- **July 5, 2023 - July 19, 2023:** Initial Code Review *(Completed)*
- **July 26, 2023:** Delivery of Initial Audit Report *(Completed)*
- **August 30, 2023:** Verification Review *(Completed)*
- **August 30, 2023:** Delivery of Final Audit Report *(Completed)*
- **September 25, 2023:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Nicole Ernst, Security Researcher and Engineer
- Mukesh Jaiswal, Security Researcher and Engineer
- Ahmad Jawid Jamiulahmadi, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the NEOKingdom DAO Smart Contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- NEOkingdom DAO contracts:
  https://github.com/NeokingdomDAO/contracts

Specifically, we examined the Git revision for our initial review:

- `3be9557ead0c8694d43caaf88591622a666211c6`

For the verification, we examined the Git revision:

- `d139cd81b922490a7d64f561db311a607e0d4478`

For the review, this repository was cloned for use during the audit and for reference in this report:

- NEOKingdom-Smart-Contracts:
  https://github.com/LeastAuthority/Neokingdom-Smart-Contracts

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Documentation: https://github.com/NeokingdomDAO/contracts/blob/main/README.md#documentation
- Brief Overview: https://vimeo.com/744685086?embedded=true&source=vimeo_logo&owner=182413982

In addition, this audit report references the following documents:

- NatSpec Format: https://docs.soliditylang.org/en/v0.8.21/natspec-format.html
- Errors and the Revert Statement: https://docs.soliditylang.org/en/v0.8.21/contracts.html#errors-and-the-revert-statement

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) or security exploits that would impact the intended use of the smart contracts or disrupt their execution;
- Vulnerabilities in the smart contracts' code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of the NEOKingdom DAO smart contract suite, which is a blockchain based, EVM compatible governance solution that allows its users to be legal shareholders of a decentralized autonomous organization (DAO) and participate in its governance, management, as well as operations.

Overall, we found that security has been taken into consideration in the design of the NEOKingdom DAO as demonstrated by appropriate implementation of access controls and re-entrancy measures. However, we identified an Issue in the design of the system whereby the excess authority of the DAO admin could allow them to acquire extra voting power (Issue D). Furthermore, we identified Issues in the implementation that could lead to unintended behavior (Issue A, Issue C), in addition to an Issue that could result in the loss of user funds (Issue B).

Code Quality

The code is well-organized and generally adheres to best practice. However, our team identified several areas of improvement that would increase the overall quality of the code. We found that using custom errors and removing redundant checks can improve gas efficiency (Suggestion 7, Suggestion 4). In addition, we identified instances of missing checks, which could result in unintended behavior (Suggestion 3).

Tests

Our team found sufficient test coverage of the smart contracts has been implemented.

Documentation

The project documentation provided for this review provides a generally sufficient overview of the system and its intended behavior. However, we recommend updating the documentation to reflect the current state of the codebase (Suggestion 1).

Code Comments

There are insufficient code comments describing security-critical components and functions in the codebase. We recommend improving code comments (Suggestion 1).

Scope

The scope of this review was sufficient and included all security-critical components and functionality of the smart contracts.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Deposited Tokens Can Be Redeemed	Resolved
Issue B: Unsettled Deposits Can Be Locked	Resolved
Issue C: Missing Modifier Preventing the Update of Non-Existent Resolutions	Resolved
Issue D: The Status of InternalMarket or ShareholderRegistry Can Be Set to Contributor Status	Partially Resolved
Issue E: settleTokens Function Mints Extra NEOK Tokens to the GovernanceToken Smart Contract (Known Issue)	Resolved
Suggestion 1: Improve Code Comments and Update the Documentation	Resolved
Suggestion 2: Add Check To Verify Token Transfer Return Value	Resolved
Suggestion 3: Add Zero Address Checks	Resolved
Suggestion 4: Remove Redundant Checks	Resolved
Suggestion 5: Use an Updated and Non-Floating Pragma Version Consistently Across the Project	Resolved
Suggestion 6: Implement the Appropriate Interface	Unresolved
Suggestion 7: Use Custom Error To Output Arguments in Error Messages	Unresolved

Issue A: Deposited Tokens Can Be Redeemed

Location

contracts/InternalMarket/InternalMarket.sol#L71-L73

contracts/GovernanceToken/GovernanceToken.sol#L205-L207

contracts/GovernanceToken/GovernanceToken.sol#L358-L370

contracts/RedemptionController/RedemptionController.sol#L55-L60

Synopsis

It is possible to redeem tokens bought from the internal or external markets, and/or tokens with an expired redemption period. Tokens bought from the internal market can be withdrawn after being offered to the internal market for seven days. After this period, these tokens can be redeposited. NEOK tokens bought from the external market can be simply deposited.

Deposited tokens can be settled after a specific period by calling the `settleTokens` function in the `GovernanceToken` smart contract. In a series of function calls, new tokens are minted to the `depositor` address and, eventually, the `afterMint` function in the `RedemptionController` smart contract will be called. This function adds the minted amount to the depositor's mint budget – the amount which can be redeemed.

Contributors could continuously withdraw and deposit the same tokens to gain unlimited redemption balance. Although this contributor would not be able to redeem more than their `GovernanceTokens` holdings balance, it enables the redemption of tokens without offering them to the internal market first. Contributors could redeem tokens received from internal or external sources easily, and activate their expired tokens by periodically withdrawing and depositing the same tokens.

Impact

This Issue enables circumventing intended restrictions on the NOEKGov token redemption, and bypassing the internal market, which contradicts the business requirements of the system.

Preconditions

In order for this Issue to occur, the user should at least be a contributor.

Mitigation

Mitigating this Issue requires further consideration and could require breaking changes. We recommend preventing the `settleTokens` function from eventually calling the `afterMint` function by adding appropriate checks.

An alternative option is to mint the tokens to one of the smart contracts – rather than the depositor's address – then transfer the amount from the smart contract to the depositor. For example, one of the smart contracts could be the `InternalMarket` smart contract.

Status

The NEOKingdom DAO team has stopped calling the `afterMint` function from the `_afterTokenTransfer` function. Instead, the function is currently called from the `mint` function inside the `GovernanceToken` smart contract, thus preventing the calling of the `afterMint` function when tokens are being settled.

Verification

Resolved.

Issue B: Unsettled Deposits Can Be Locked

Location

contracts/InternalMarket/InternalMarket.sol#L71-L73

contracts/GovernanceToken/GovernanceToken.sol#L205-L207

Synopsis

It is possible for a user to deposit (wrap) zero external tokens using the `deposit` function in the `InternalMarket` smart contract. When settling tokens using the `settleTokens` function in the `GovernanceToken` smart contract, the internal function `_settleTokens` uses an iteration to settle a list of unsettled deposits. In each iteration, it checks if the amount to be settled is greater than zero. If the condition fails, the iteration will stop.

Impact

Due to the aforementioned check, if a user deposits zero NEOK tokens in addition to their previously unsettled tokens, previously unsettled tokens will be locked.

Preconditions

The user must add a zero token deposit to other unsettled token deposits.

Technical Details

```
function _settleTokens(address from) internal virtual {
        for (uint256 i = depositedTokens[from].length; i > 0; i--) {
            DepositedTokens storage tokens = depositedTokens[from][i - 1];
            if (block.timestamp >= tokens.settlementTimestamp) {
                if (tokens.amount > 0) {
                    super._mint(from, tokens.amount);
                    tokens.amount = 0;
                } else {
                    break;
                }
            }
        }
    }
```

Remediation

We recommend preventing zero deposit amounts by adding a check in the `deposit` function.

Status

The NEOKingdom DAO team has added the check for zero amounts as suggested.

Verification

Resolved.

Issue C: Missing Modifier Preventing the Update of Non-Existent Resolutions

Location

`contracts/ResolutionManager/ResolutionManagerBase.sol#L257-L264`

`contracts/ResolutionManager/ResolutionManagerBase.sol#L119-L126`

Synopsis

The function `_updateResolution` does not have the `exists` modifier to prevent the update of resolutions that do not exist.

Impact

This Issue could result in the creation of another resolution rather than updating an existing one.

Preconditions

This Issue is likely if the `resolutionId` passed to the function does not exist.

Remediation

We recommend adding the `exists` modifier to the function to verify whether the resolution exists.

Status

The NEOKingdom DAO team has added the modifier to check for `resolutionId`.

Verification

Resolved.

Issue D: The Status of InternalMarket or ShareholderRegistry Can Be Set to Contributor Status

Location

`contracts/ShareholderRegistry/ShareholderRegistry.sol#L65-L70`

`contracts/Voting/Voting.sol#L147-L152`

Synopsis

The `RESOLUTION_ROLE`, which is currently controlled by a Multi-Sig, can be set by the `DEFAULT_ADMIN_ROLE` to any address. Additionally, the `RESOLUTION_ROLE` can set the status of `InternalMarket` or `ShareholderRegistry` to the `CONTRIBUTOR_STATUS` and then delegate their

potential voting power to an arbitrary contributor using the `delegateFrom` function in the `Voting` smart contract.

Impact

The `ShareholderRegistry` smart contract's shares and `InternalMarket`'s governance tokens are counted as voting power. Consequently, these voting powers might potentially be misused since they can be delegated to any contributor.

Remediation

Even though the setting of the aforementioned state changes may go through a resolution process, we still recommend that the smart contract self-guard against this by preventing the setting of a status for the aforementioned contracts.

Status

The NEOKingdom DAO team has added a check to prevent setting a status if the address is a contract. However, this fix will prevent the setting of a status for valid contributors if the contributor address is a Multi-Sig address.

Verification

Partially Resolved.

Issue E: settleTokens Function Mints Extra NEOK Tokens to the GovernanceToken Smart Contract (Known Issue)

Location

`contracts/GovernanceToken/GovernanceToken.sol#L366-L378`

`contracts/GovernanceToken/GovernanceTokenBase.sol#L60`

Synopsis

The `settleTokens` function in the `GovernanceToken` smart contract can be used to settle deposited tokens, after waiting for a specific period, once the deposit is complete.

When the function is called after the specific waiting period, it internally calls the `_mint` function in the `GovernanceTokenBase` smart contract to mint new NEOKGov tokens to the depositor. However, this function mints new external (NEOK) tokens to the `GovernanceToken` smart contract, additionally to the NEOKGov tokens minted to the depositor. This is unnecessary, extra minting since NEOK tokens had already been deposited.

Impact

Even though the tokens are not minted to the depositor address but to the GovernanceToken smart contract instead, it will break the 1:1 ratio of NEOK and NEOKGov tokens, resulting in more NEOK tokens in the circulation.

Preconditions

This Issue can occur when external tokens are deposited to be substituted with NEOKGov tokens.

Status

The NEOKingdom DAO team found this Issue during the audit and remediated it by refraining from calling the `_mint` function in the `GovernanceTokenBase` smart contract and, instead, minting NEOKGov tokens directly to the depositor in the `settleTokens` function.

Verification

Resolved.

Suggestions

Suggestion 1: Improve Code Comments and Update the Documentation

Location

`contracts/ShareholderRegistry/ShareholderRegistryBase.sol#L79-L92`

`contracts/Voting/Voting.sol#L97`

Synopsis

There is high-level documentation describing how the various components in the NEOKingdom DAO smart contracts interact with each other. However, there were insufficient code comments explaining the role of individual functions and variables.

Additionally, in the documentation and in the code comments, it is mentioned that the shareholder, investor, contributor, and managing board have, respectively, increasing levels of privileges. However, in the `_isAtLeast` function in the `ShareholderRegistryBase` smart contract, the shareholder and investor are considered to be the same when determining the account status.

Furthermore, in the comments, it is mentioned that the `afterTokenTransfer` function can only be called by the `GovernanceToken` contract. However, it can actually be called by both the `GovernanceToken` contract and the `shareholderRegistry` contract.

Mitigation

We recommend adding more code comments while adhering to the NatSpec standard and updating the documentation and code comments to reflect the implementation.

Status

The NEOKingdom DAO team prioritized documenting the public interface of their smart contracts and also added comments to other components with complex logic.

Verification

Resolved.

Suggestion 2: Add Check To Verify Token Transfer Return Value

Location

`GovernanceTokenBase.sol#L72`

`GovernanceToken.sol#L345`

Synopsis

The return value of the token transfer is not checked consistently. While the token transfer's return value in the `InternalMarket` contract is checked, the `_wrap` and `_unwrap` functions of the `GovernanceToken` contract lack these checks.

Mitigation

We recommend adding checks to verify the return value of the token transfer.

Status

The NEOKingdom DAO team has added the checks for the token transfer as suggested.

Verification

Resolved.

Suggestion 3: Add Zero Address Checks

Location

Examples (non-exhaustive):

`contracts/ResolutionManager/ResolutionManager.sol#L23-L27`

`contracts/InternalMarket/InternalMarket.sol#L26-L27`

Synopsis

There are many input parameters in the smart contracts with missing zero address checks validating the correctness of those parameters to prevent incorrectly set values.

Mitigation

We recommend implementing missing zero address checks in the smart contracts.

Status

The NEOKingdom DAO team has added zero address checks as suggested.

Verification

Resolved.

Suggestion 4: Remove Redundant Checks

Location

`contracts/ResolutionManager/ResolutionManager.sol#L170-L176`

`contracts/ResolutionManager/ResolutionManager.sol#L185-L191`

`contracts/ResolutionManager/ResolutionManager.sol#L212-L218`

Synopsis

The referenced checks in the `ResolutionManager` smart contract functions are also implemented in their corresponding internal functions in the `ResolutionManagerBase` smart contract, resulting in an unnecessary consumption of gas.

Mitigation

We recommend keeping only one of the checks and removing the other one to save gas.

Status

The NEOKingdom DAO team has removed the redundant checks as suggested.

Verification

Resolved.

Suggestion 5: Use an Updated and Non-Floating Pragma Version Consistently Across the Project

Location

Examples (non-exhaustive):

`contracts/GovernanceToken/GovernanceToken.sol#L2`

`contracts/NeokingdomToken/NeokingdomToken.sol#L2`

`contracts/extensions/DAORoles.sol#L3`

Synopsis

The version pragmas on the contracts are floating and inconsistent. Some are `^0.8.16`, while others are `^0.8.9` or `^0.8.0`. Compiling with different compiler versions may cause conflicts and unexpected results and possibly lead to the smart contracts being deployed with an unintended compiler version, which could result in unexpected behavior.

Mitigation

In order to maintain consistency and to prevent unexpected behavior, we recommended that the Solidity compiler version be pinned by removing "^", updated to one of the latest versions, and used consistently across the system.

Status

The NEOKingdom DAO team has implemented the remediation as recommended.

Verification

Resolved.

Suggestion 6: Implement the Appropriate Interface

Location

`contracts/NeokingdomToken/INeokingdomToken.sol`

`contracts/NeokingdomToken/NeokingdomToken.sol`

Synopsis

The `NeokingdomToken` smart contract, by design, relies on the former interface `INeokingdomToken`. However, the interface does not appear to be implemented, which can prevent the interaction with the smart contract.

Mitigation

We recommend implementing the following:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.16;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";

import "@openzeppelin/contracts/access/AccessControl.sol";

import "./INeokingdomToken.sol";

 ...

contract NeokingdomToken is ERC20, ERC20Burnable, AccessControl,
INeokingdomToken {

    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

 ...
```

Status

The NEOKingdom DAO team has not addressed this suggestion. Currently, the smart contract and interface are defined identically. However, the interface has not been explicitly implemented yet. Hence, if the `NeokingdomToken` contract and/or the `INeokingdomToken` interface are/is updated in such a way that renders them different, this may result in unintended behavior. We recommend implementing the appropriate interface for the smart contract as suggested above.

Verification

Unresolved

Suggestion 7: Use Custom Error To Output Arguments in Error Messages

Location

[contracts/extensions/HasRole.sol#L28-L37]

Synopsis

The code referenced above includes arguments in the error messages but is complex and not gas efficient.

Mitigation

We recommend using custom errors to output data in error messages and to save gas.

Status

The NEOKingdom DAO team stated that they implemented the error message in this manner in an attempt to make all the error messages in the codebase consistent with the error message utilized in `AccessControl.sol` from OpenZeppelin.

Verification

Unresolved

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.