Election

Smart Contract Audit Report
Prepared for iAM

Date Issued:	Apr 25, 2022
Project ID:	AUDIT2022012
Version:	v2.0
Confidentiality Level:	Public

Report Information

Project ID	AUDIT2022012
Version	v2.0
Client	iAM
Project	Election
Auditor(s)	Natsasit Jirathammanuwat Puttimet Thammasaeng
Author(s)	Puttimet Thammasaeng
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	Apr 25, 2022	Update the scope URLs.	Puttimet Thammasaeng
1.0	Apr 19, 2022	Full report	Puttimet Thammasaeng

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by iAM, Inspex team conducted an audit to verify the security posture of the Election smart contracts on Feb 22, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Election smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 2 very low-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that Election smart contracts have high-level protections in place to be safe from most attacks.

This smart contract passes Inspex's security verification standard, and is trustworthy.

Approved by Inspex on Apr 19, 2022

inspex CYBERSECURITY PROFESSIONAL SERVICE

PASS

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Election contracts provide a voting mechanism with the ERC20 token, the users can submit the number of tokens for voting in an election. The token that users have used for voting can be locked or burned depending on the voting type. Moreover, the voting mechanism implements the commit-reveal scheme which users cannot view the result before the voting ends.

Scope Information:

Project Name	Election
Website	https://www.bnk48.com/
Smart Contract Type	Ethereum Smart Contract
Chain	TKX Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Feb 22, 2022
Reassessment Date	Feb 28, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 4cc7b49156a2099a2cc506d5c2961abbe0fd2372)

Contract	Location (URL)
AdminManage	https://github.com/inspex-archive/iAM_Election/blob/4cc7b49156/admin/adminManage.sol
ElectionFactory	https://github.com/inspex-archive/iAM_Election/blob/4cc7b49156/election/ElectionFactory.sol
ElectionPoll	https://github.com/inspex-archive/iAM_Election/blob/4cc7b49156/election/ElectionPoll.sol
ElectionVerify	https://github.com/inspex-archive/iAM_Election/blob/4cc7b49156/election/ElectionVerify.sol

Reassessment: (Commit: -)

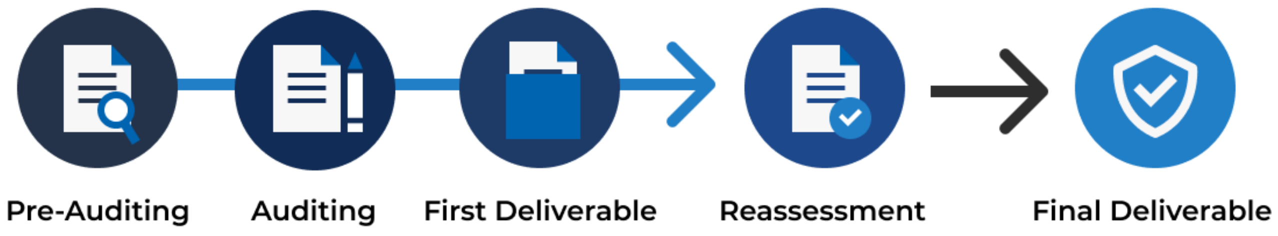
Contract	Location (URL)
AdminManage	https://scan.tokenx.finance/address/0x30D977cD663bA6B73794E588F98196d90458cC06/contracts
ElectionFactory	https://scan.tokenx.finance/address/0x3D536f049DD38D5E1f6feb6BBcD1e304e24c2deD/contracts
ElectionPoll	https://scan.tokenx.finance/address/0xB03045aDD4ce13Ceb1551943308bdb7cdc4aA5D3/contracts
ElectionVerify	https://scan.tokenx.finance/address/0x3FC3661fB422dD9eD172bcF4797638865E1f0DD7/contracts

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Centralized Control of State Variable
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication

Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

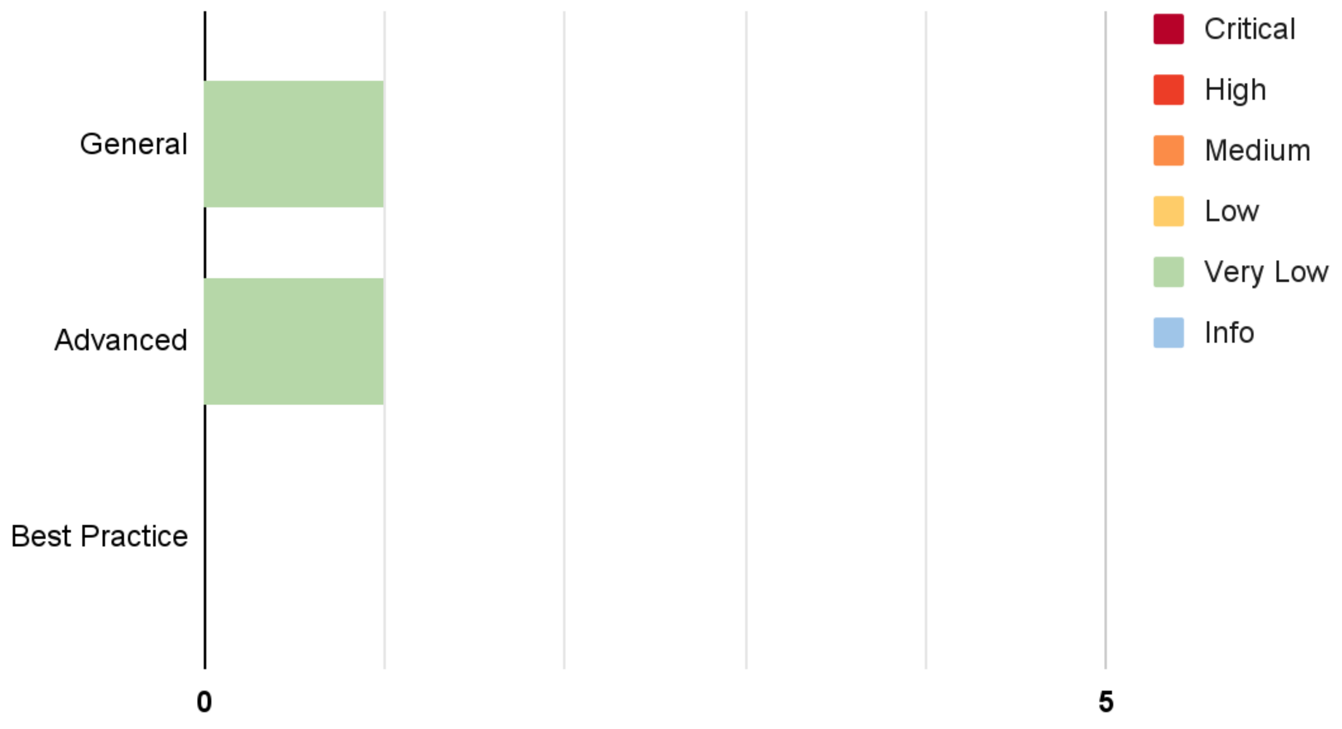
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Likelihood / Impact	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

4. Summary of Findings

From the assessments, Inspex has found 2 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Improper removeSuperAdmin() Function Implementation	Advanced	**Very Low**	Resolved
IDX-002	Insufficient Logging for Privileged Functions	General	**Very Low**	Resolved

* The mitigations or clarifications by iAM can be found in Chapter 5.

5. Detailed Findings Information

5.1. Improper removeSuperAdmin() Function Implementation

ID	IDX-001
Target	AdminManage
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Very Low **Impact:** Low The super admin can prevent himself from being removed from the super admin role. **Likelihood:** Low This issue is very unlikely to be exploited since the remove admin function requires the caller to have the super admin role. Moreover, the pending removed admin must monitor the mempool and call the `removeSuperAdmin()` function when the `tmp.kickVote` is set.
Status	Resolved The iAM team has resolved this issue as suggested by validating the function is not called by the pending removed super admin.

5.1.1. Description

Any super admin can call the `removeSuperAdmin()` function to create a vote for removing the other super admin. The first function call will set the `tmp.kickVote` state to the pending removal of the admin address. Then the second function call will remove the pending removed admin address from the super admin role and clear the `tmp.kickVote` state.

However, the `tmp.kickVote` state can be set again without waiting for the deadline if the `_superAdmin` parameter is not equal to the `tmp.kickVote` state as shown below in lines 194-198.

AdminManage.sol

```
189 function removeSuperAdmin(address _superAdmin) external onlySuperAdmin {
190   require(
191     superAdmin[_superAdmin].next != address(0),
192     'AdminManage [removeSuperAdmin]: require superAdmin wallet at arg[0]'
193   );
194   if (block.number > tmp.deadline || _superAdmin != tmp.kickVote) {
195     tmp = Vote({lastVoter: msg.sender, kickVote: _superAdmin, deadline:
    block.number + deadline});
196
197     return;
198   }
```

```

  // ====  vote to same address ====
  // require vote 1 vote 1 address
  require(msg.sender != tmp.lastVoter, "AdminManage [removeSuperAdmin]: can't use the same address to vote");

  _removeSuperAdmin(_superAdmin);

  // empty tmp
  tmp = Vote({lastVoter: address(0), kickVote: address(0), deadline: block.number});
}
```

This results in the pending removed admin being able to prevent himself from being removed by calling the `removeSuperAdmin()` function and setting the `tmp.kickVote` state to the other super admin.

5.1.2. Remediation

Inspex suggests validating whether the `removeSuperAdmin()` function is not called by the current `tmp.kickVote` address, for example in lines 194-197:

AdminManage.sol

```
189 function removeSuperAdmin(address _superAdmin) external onlySuperAdmin {
190   require(
191     superAdmin[_superAdmin].next != address(0),
192     'AdminManage [removeSuperAdmin]: require superAdmin wallet at arg[0]'
193   );
194   require(
195     block.number > tmp.deadline || msg.sender != tmp.kickVote,
196     'AdminManage [removeSuperAdmin]: tmp.kickVote cannot remove any super admin role'
197   );
198   if (block.number > tmp.deadline || _superAdmin != tmp.kickVote) {
199     tmp = Vote({lastVoter: msg.sender, kickVote: _superAdmin, deadline: block.number + deadline});
200 
201     return;
202   }
```

5.2. Insufficient Logging for Privileged Functions

ID	IDX-002
Target	ElectionPoll
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved The iAM team has resolved this issue as suggested by emitting the event on the affected functions.

5.2.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can change the voting period by executing the `editPollVoteTime()` function without emitting any event.

ElectionPoll.sol

```
function editPollVoteTime(uint256 _startBlock, uint256 _endBlock) external onlyAdmin onlyBeforeStart {
  require(endBlock > _startBlock, 'ElectionPoll [editPollVoteTime]: endBlock should more then startBlock.');

  startBlock = _startBlock;
  endBlock = _endBlock;
}
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function
ElectionPoll.sol (L: 110)	ElectionPoll	editPollVoteTime()
ElectionPoll.sol (L: 117)	ElectionPoll	directClosePoll()

5.2.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

ElectionPoll.sol

```
event EditPollVoteTimeLog(uint256 _startBlock, uint256 _endBlock);
function editPollVoteTime(uint256 _startBlock, uint256 _endBlock) external
onlyAdmin onlyBeforeStart {
  require(endBlock > _startBlock, 'ElectionPoll [editPollVoteTime]: endBlock
should more then startBlock.');

  startBlock = _startBlock;
  endBlock = _endBlock;
  emit EditPollVoteTimeLog(_startBlock, _endBlock);
}
```

6. Appendix

6.1. About Inspex

CYBERSECURITY PROFESSIONAL SERVICE

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] "OWASP Risk Rating Methodology." [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE