Admin & Poll

Smart Contract Audit Report
Prepared for iAM

Date Issued:	Jan 19, 2022
Project ID:	AUDIT2021042
Version:	v1.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021042
Version	v1.0
Client	iAM
Project	Admin & Poll
Auditor(s)	Suvicha Buakhom Peeraphut Punsuwan Darunphop Pengkumta
Author(s)	Peeraphut Punsuwan
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jan 19, 2022	Full report	Peeraphut Punsuwan

Contact Information

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

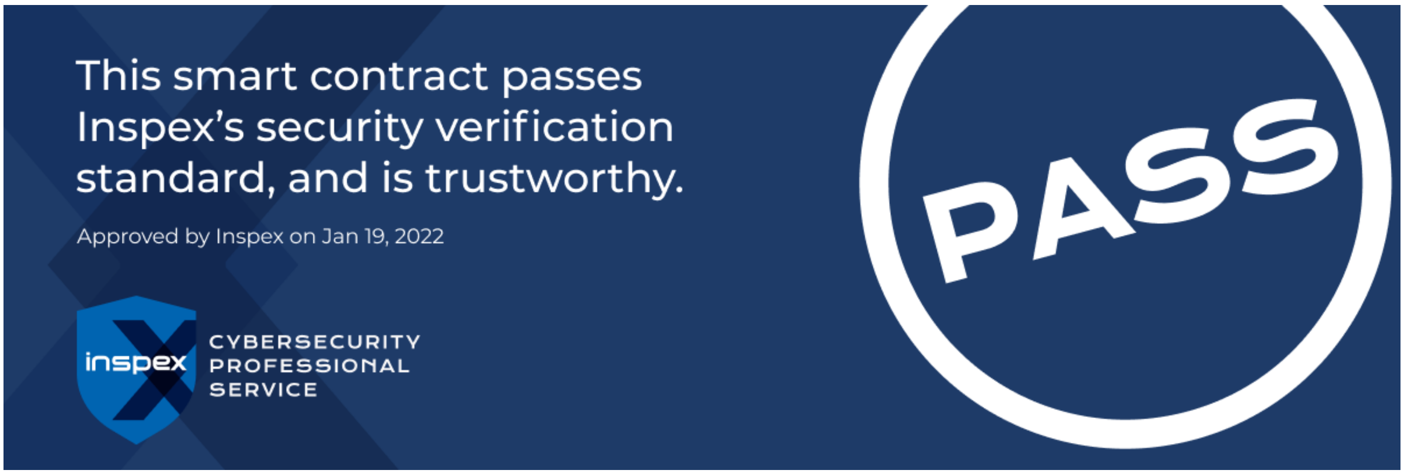
Table of Contents

1. Executive Summary

As requested by iAM, Inspex team conducted an audit to verify the security posture of the Admin & Poll smart contracts between Jan 5, 2022 and Jan 6, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Admin & Poll 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 1 low, 2 very low and 3 info-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 Admin & Poll 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 Jan 19, 2022

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

Admin & Poll are the contracts of the iAM platform. The Poll contracts have a voting mechanism with the ERC20 token that users hold for a vote. The users can vote with multi-vote options depending on the poll type. Moreover, the Admin contract is used for managing the admin members with the dual control mechanism which requires more than one admin to execute the privileged functions.

Scope Information:

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

Audit Information:

Audit Method	Whitebox
Audit Date	Jan 5, 2022 - Jan 6, 2022
Reassessment Date	Jan 17, 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: 0f8bb7a04c439286a5dffa223c3b30265acb4b5c)

Contract	Location (URL)
AdminManage	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/admin/adminManage.sol
BigOwner	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/admin/BigOwner.sol
Poll	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/poll/Poll.sol
PollFactory	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/poll/PollFactory.sol
MultipleTransferProxy	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/MultipleTransferProxy.sol
TokenFactory	https://github.com/inspex-archive/iAM-Admin-Poll-Token/blob/0f8bb7a04c/TokenFactory.sol

Please note that the iAM smart contracts were initially in iAM's private repository. The files have been uploaded to Inspex's archive repository for public access.

Reassessment:

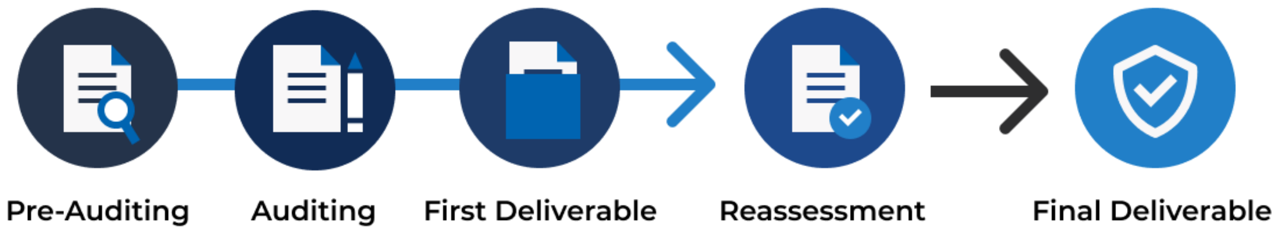
Contract	Location (URL)
AdminManage	https://scan.tokenx.finance/address/0x9f3a8b4Bd35B310f07774944b8B9d73432BE2a4F/contracts
BigOwner	https://scan.tokenx.finance/address/0x4b06C75263D7D3c0157864FA980D05003B06Fe5b/contracts
Poll	https://scan.tokenx.finance/address/0x129C1e67596123Cd21C1FaaE603f2e6dE152A928/contracts
PollFactory	https://scan.tokenx.finance/address/0xa8b0F8ca2c708f6dF118b92902d07311586587bb/contracts
MultipleTransferProxy	https://scan.tokenx.finance/address/0x129C1e67596123Cd21C1FaaE603f2e6dE152A928/contracts
TokenFactory	https://scan.tokenx.finance/address/0x87328D24F9a61c64b66CD978B169b3955488d0d1/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
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

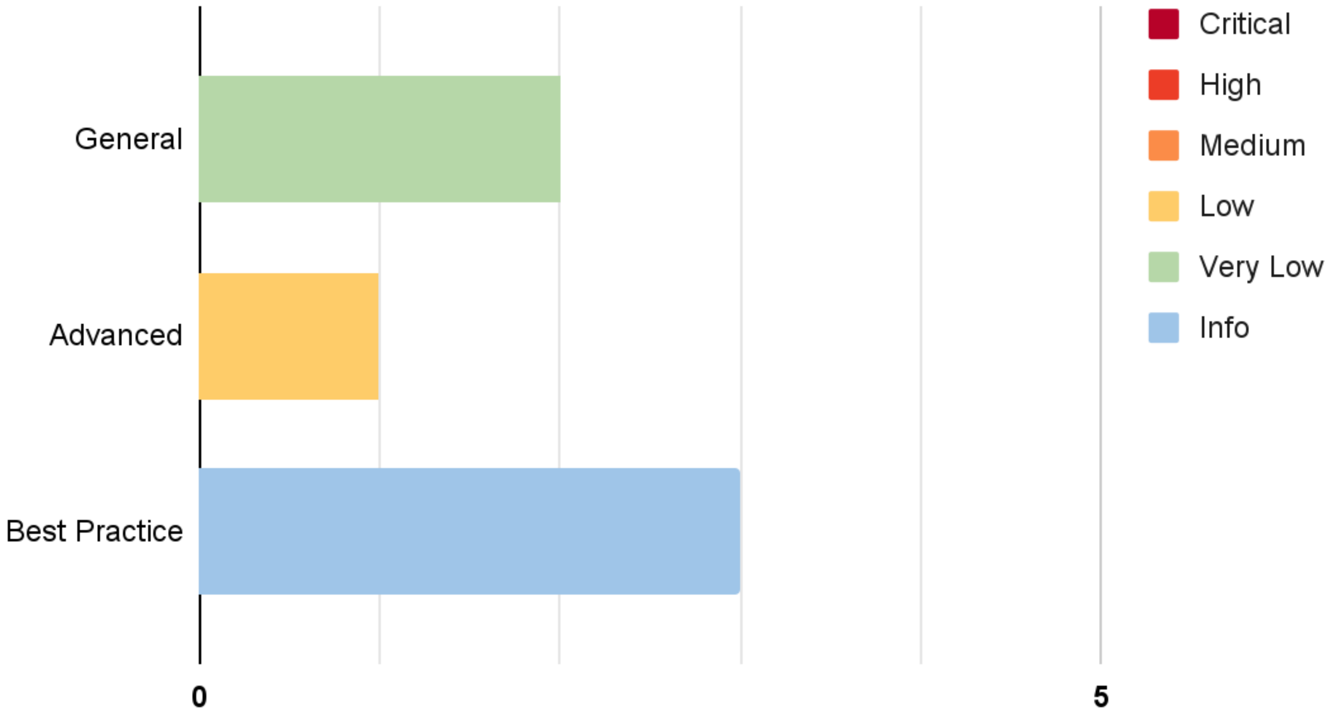
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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 6 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	Missing Approval Validation	Advanced	**Low**	**Resolved**
IDX-002	Insufficient Logging for Privileged Functions	General	**Very Low**	**Resolved**
IDX-003	Outdated Solidity Compiler Version	General	**Very Low**	**Resolved**
IDX-004	Unnecessary Assert Statement	Best Practice	**Info**	**Resolved**
IDX-005	Improper Function Visibility	Best Practice	**Info**	**Resolved**
IDX-006	Incorrect Logging parameter for Privileged Functions	Best Practice	**Info**	**Resolved**

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

5. Detailed Findings Information

5.1. Missing Approval Validation

ID	IDX-001
Target	BigOwner
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Medium One of the admins in the `BigOwner` contract can remove other admins and add new admins without any approval from other admins. The other admins will lose their ownership of the contract through the action from just one admin. **Likelihood:** Low Only the admin can execute the `setPendingAdmin()` function. The newly gained privilege has the same level of the privilege as the admin who executes the function, so the motivation for the attack is not high.
Status	Resolved The iAM team has resolved this issue by validating the `pendingAdminHashApprove` state to be true in the `acceptAdmin()` function.

5.1.1. Description

The `BigOwner` contract is designed to have three consecutive steps to replace an admin: `setPendingAdmin()`, `approvePendingAdmin()`, and `acceptAdmin()` functions.

The workflow for replacing an admin starts from `setPendingAdmin()` function. The function is used for defining an admin user that is going to be replaced and a normal user (non-admin user) that is going to be an admin. The function will also set `pendingAdminHashApprove` state to `false`. The `pendingAdminHashApprove` state requires another function to approve the proposal by setting it to `true`.

BigOwner.sol

```
114 function setPendingAdmin(address pendingAdmin_, address pendingRemoveAdmin_)
    external onlyOwner {
115     // allows one time setting of admin for deployment purposes
116     require(admin[pendingRemoveAdmin_] == true, 'BigOwner::setPendingAdmin:
    pendingRemoveAdmin should be admin.');
117     require(admin[pendingAdmin_] == false, 'BigOwner::setPendingAdmin:
    pendingAdmin should not be admin.');
118
```

```
    bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
    pendingAdminHashSubmitter = msg.sender;
    pendingAdminHashSubmitBlock = getBlockNumber();
    pendingAdminHashApprove = false;
    pendingAdminHash = txHash;

    emit PendingAdmin(pendingAdmin_, pendingRemoveAdmin_);
}
```

The `approvePendingAdmin()` function is used for approving the admin replacing a proposal that has been initiated from the `setPendingAdmin()` function by another admin. This function needs to be called from another admin to approve the proposal. Then it will set the `pendingAdminHashApprove` state to `true`.

BigOwner.sol

```
function approvePendingAdmin(address pendingAdmin_, address
pendingRemoveAdmin_) external onlyOwner returns (bool) {
    bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
    require(txHash == pendingAdminHash, 'BigOwner::approvePendingAdmin:
Argument not match to pendingAdminHash.');
    require(
        msg.sender != pendingAdminHashSubmitter,
        'BigOwner::approvePendingAdmin: Call must not come from
pendingAdminHashSubmitter.'
    );

    pendingAdminHashApprove = true;

    emit ApprovePendingAdmin(pendingAdmin_, pendingRemoveAdmin_);
    return true;
}
```

The `acceptAdmin()` function is used for accepting the admin position by a normal user. Considering the last two functions, the proposal needs to be approved first and the state of approval that represents by `pendingAdminHashApprove` state should be `true`.

BigOwner.sol

```
function acceptAdmin(address pendingAdmin_, address pendingRemoveAdmin_) public
{
    bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
    require(txHash == pendingAdminHash, 'BigOwner::acceptAdmin: Argument not
match pendingAdminHash.');
    require(msg.sender == pendingAdmin_, 'BigOwner::acceptAdmin: Call must come
from newAdmin.');
    require(
        pendingAdminHashSubmitBlock + PANDING_BLOCK >= getBlockNumber(),
75            "BigOwner::acceptAdmin: PendingAdmin hasn't surpassed pending time."
76        );
77
78        admin[pendingAdmin_] = true;
79        admin[pendingRemoveAdmin_] = false;
80
81        uint256 removeAdminIndex = 0;
82        bool isFound = false;
83        for (uint256 i = 0; i < adminList.length; i++) {
84            if (adminList[i] == pendingRemoveAdmin_) {
85                removeAdminIndex = i;
86                isFound = true;
87                break;
88            }
89        }
90        assert(isFound);
91
92        adminList[removeAdminIndex] = pendingAdmin_;
93
94        pendingAdminHashSubmitter = address(0);
95        pendingAdminHash = '';
96
97        emit NewAdmin(pendingAdmin_, pendingAdmin_);
98    }
```

But in this function, `acceptAdmin()`, doesn't check the state of `pendingAdminHashApprove` being set to `true` yet. So, the admin can propose a new admin position to a user, and the user can immediately accept the position without any approval from another admin. Potentially, an admin can replace the rest of admins with their normal user accounts.

5.1.2. Remediation

Inspex suggests validating the `pendingAdminHashApprove` state to be `true` in the `acceptAdmin()` function, for example: adding a validation of `pendingAdminHashApprove` at line 77.

BigOwner.sol

```
69    function acceptAdmin(address pendingAdmin_, address pendingRemoveAdmin_) public
      {
70        bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
71        require(txHash == pendingAdminHash, 'BigOwner::acceptAdmin: Argument not
      match pendingAdminHash.');
72        require(msg.sender == pendingAdmin_, 'BigOwner::acceptAdmin: Call must come
      from newAdmin.');
73        require(
74            pendingAdminHashSubmitBlock + PANDING_BLOCK >= getBlockNumber(),
75            "BigOwner::acceptAdmin: PendingAdmin hasn't surpassed pending time."
```

```
    );
    require(pendingAdminHashApprove, "BigOwner::acceptAdmin: This action needs
to be approved.");

    admin[pendingAdmin_] = true;
    admin[pendingRemoveAdmin_] = false;

    uint256 removeAdminIndex = 0;
    bool isFound = false;
    for (uint256 i = 0; i < adminList.length; i++) {
        if (adminList[i] == pendingRemoveAdmin_) {
            removeAdminIndex = i;
            isFound = true;
            break;
        }
    }
    assert(isFound);

    adminList[removeAdminIndex] = pendingAdmin_;

    pendingAdminHashSubmitter = address(0);
    pendingAdminHash = '';

    emit NewAdmin(pendingAdmin_, pendingAdmin_);
}
```

5.2. Insufficient Logging for Privileged Functions

ID	IDX-002
Target	AdminManage BigOwner
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 by emitting events for the execution of privileged 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 to the platform.

For example, the owner can modify the admins by executing `addAdmin()` function in the `AdminManage` contract, and no event is emitted.

adminManage.sol

```
function addAdmin(address _newAdmin) public onlySuperAdmin {
    require(
        _newAdmin != address(0) && _newAdmin != GENESIS_ADDRESS,
        'AdminManage [addAdmin]: Invalid owner address provided'
    );
    require(admin[_newAdmin].next == address(0), 'AdminManage [addAdmin]:
Address is already a admin');

    address latest = admin[GENESIS_ADDRESS].prev;
    admin[latest].next = _newAdmin;

    admin[_newAdmin].prev = latest;
    admin[_newAdmin].next = GENESIS_ADDRESS;
    admin[GENESIS_ADDRESS].prev = _newAdmin;
    adminCount++;
}
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function	Modifier
adminManage.sol (L:83)	AdminManage	addAdmin()	onlySuperAdmin
adminManage.sol (L:99)	AdminManage	addSuperAdmin()	onlySuperAdmin
adminManage.sol (L:115)	AdminManage	removeAdmin()	onlySuperAdmin
adminManage.sol (L:143)	AdminManage	removeSuperAdmin()	onlySuperAdmin
BigOwner.sol (L:65)	BigOwner	setDeadline()	onlyOwner

5.2.2. Remediation

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

adminManage.sol

```
event AddAdmin(address _newAdmin);
function addAdmin(address _newAdmin) public onlySuperAdmin {
    require(
        _newAdmin != address(0) && _newAdmin != GENESIS_ADDRESS,
        'AdminManage [addAdmin]: Invalid owner address provided'
    );
    require(admin[_newAdmin].next == address(0), 'AdminManage [addAdmin]: 
Address is already a admin');

    address latest = admin[GENESIS_ADDRESS].prev;
    admin[latest].next = _newAdmin;

    admin[_newAdmin].prev = latest;
    admin[_newAdmin].next = GENESIS_ADDRESS;
    admin[GENESIS_ADDRESS].prev = _newAdmin;
    adminCount++;
    emit AddAdmin(_newAdmin);
}
```

5.3. Outdated Solidity Compiler Version

ID	IDX-003
Target	AdminManage BigOwner Poll PollFactory MultipleTransferProxy TokenFactory
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves. **Likelihood:** Low From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts.
Status	Resolved The iAM team has resolved by upgrading the Solidity compiler version to 0.8.9 which currently does not have any known vulnerability.

5.3.1. Description

The solidity compiler versions declared in the smart contracts were outdated. These versions have publicly known inherent bugs[2] that may potentially be used to cause damage to the smart contracts or the users of the smart contracts.

adminManage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.4;
```

The following table contains all targets which the outdated compiler version is declared.

Contract	Version
AdminManage	=0.8.4
BigOwner	=0.8.4
Poll	=0.8.4

PollFactory	=0.8.4
MultipleTransferProxy	=0.8.4
TokenFactory	=0.8.4

5.3.2. Remediation

Inspex suggests upgrading the solidity compiler to the latest stable version[3]. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.11.

adminManage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.11;
```

Please note that some dependencies that are imported in each contract require a specific compiler version, please verify their compatibility before upgrading them.

5.4. Unnecessary Assert Statement

ID	IDX-004
Target	PollFactory
Category	Advanced Smart Contract Vulnerability
CWE	CWE-1164: Irrelevant Code
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The iAM team has resolved this issue by removing the affected assert statements that are unnecessary.

5.4.1. Description

The condition values of the assert statement in the `PollFactory` contract is always `true`. So, it is not necessary to check and can be removed to reduce the gas usage.

In line 51 of the `PollFactory.sol` is unnecessary having the assert statements, because the result of the condition is always `true`.

PollFactory.sol

```
22 function createPoll(
23     IERC20Burnable _voteToken,
24     string memory _question,
25     string memory _desc,
26     string memory _name,
27     uint256 _startBlock,
28     uint256 _endBlock,
29     uint256 _minimumToken,
30     uint256 _maximumToken,
31     bool _burnType,
32     bool _multiType,
33     string[] memory _proposals
34 ) external onlyAdmin returns (address addr) {
35     bytes memory bytecode = getContractBytecode(getAdminManage(), _voteToken,
   _question, _desc, _name);
36
37     bytes32 salt = keccak256(abi.encodePacked(block.number, msg.sender));
38
39     assembly {
        addr := create2(0, add(bytecode, 0x20), mload(bytecode), salt)
        if iszero(extcodesize(addr)) {
            revert(0, 0)
        }
    }

    Poll(addr).initialize(_startBlock, _endBlock, _minimumToken, _maximumToken,
_burnType, _multiType, _proposals);

    PollInfo memory pollInfo = PollInfo(pollCount, addr, block.timestamp);

    polls.push(pollInfo);
    assert(pollCount + 1 > pollCount);
    pollCount++;

    addressToPoll[addr] = pollInfo;

    emit PollCreated(addr, msg.sender, _name);
}
```

5.4.2. Remediation

Inspex suggests removing the affected assert statements that are unnecessary, for example:

PollFactory.sol

```
function createPoll(
    IERC20Burnable _voteToken,
    string memory _question,
    string memory _desc,
    string memory _name,
    uint256 _startBlock,
    uint256 _endBlock,
    uint256 _minimumToken,
    uint256 _maximumToken,
    bool _burnType,
    bool _multiType,
    string[] memory _proposals
) external onlyAdmin returns (address addr) {
    bytes memory bytecode = getContractBytecode(getAdminManage(), _voteToken,
_question, _desc, _name);

    bytes32 salt = keccak256(abi.encodePacked(block.number, msg.sender));

    assembly {
        addr := create2(0, add(bytecode, 0x20), mload(bytecode), salt)
        if iszero(extcodesize(addr)) {
            revert(0, 0)
        }
    }

    Poll(addr).initialize(_startBlock, _endBlock, _minimumToken, _maximumToken,
_burnType, _multiType, _proposals);

    PollInfo memory pollInfo = PollInfo(pollCount, addr, block.timestamp);

    polls.push(pollInfo);
    pollCount++;

    addressToPoll[addr] = pollInfo;

    emit PollCreated(addr, msg.sender, _name);
}
```

5.5. Improper Function Visibility

ID	IDX-005
Target	AdminManage BigOwner Poll
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The iAM team has resolved this issue by changing all affected functions' visibility to external.

5.5.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

For example, the following source code shows that the `addAdmin()` function of the `AdminManage` contract is set to public and it is never called from any internal function.

adminManage.sol

```
function addAdmin(address _newAdmin) public onlySuperAdmin {
    require(
        _newAdmin != address(0) && _newAdmin != GENESIS_ADDRESS,
        'AdminManage [addAdmin]: Invalid owner address provided'
    );
    require(admin[_newAdmin].next == address(0), 'AdminManage [addAdmin]:
Address is already a admin');

    address latest = admin[GENESIS_ADDRESS].prev;
    admin[latest].next = _newAdmin;

    admin[_newAdmin].prev = latest;
    admin[_newAdmin].next = GENESIS_ADDRESS;
    admin[GENESIS_ADDRESS].prev = _newAdmin;
    adminCount++;
}
```

Public

The following table contains all functions that have `public` visibility and are never called from any internal function.

File	Contract	Function
adminManage.sol (L:99)	AdminManage	addSuperAdmin()
adminManage.sol (L:115)	AdminManage	removeAdmin()
adminManage.sol (L:143)	AdminManage	removeSuperAdmin()
BigOwner.sol (L:69)	BigOwner	acceptAdmin()
Poll.sol (L:91)	Poll	initialize()
Poll.sol (L:150)	Poll	addProposalNames()
Poll.sol (L:155)	Poll	editProposalDesc()

5.5.2. Remediation

Inspex suggests changing all functions' visibility to `external` if they are not called from any `internal` function as shown in the following example:

adminManage.sol

```
function addAdmin(address _newAdmin) external onlySuperAdmin {
    require(
        _newAdmin != address(0) && _newAdmin != GENESIS_ADDRESS,
        'AdminManage [addAdmin]: Invalid owner address provided'
    );
    require(admin[_newAdmin].next == address(0), 'AdminManage [addAdmin]:
Address is already a admin');

    address latest = admin[GENESIS_ADDRESS].prev;
    admin[latest].next = _newAdmin;

    admin[_newAdmin].prev = latest;
    admin[_newAdmin].next = GENESIS_ADDRESS;
    admin[GENESIS_ADDRESS].prev = _newAdmin;
    adminCount++;
}
```

5.6. Incorrect Logging Parameter for Privileged Functions

ID	IDX-006
Target	BigOwner
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The iAM team has resolved this issue by changing the `pendingRemoveAdmin_` parameter value in the event from `pendingAdmin_` to `pendingRemoveAdmin_` variable.

5.6.1. Description

The `NewAdmin` event with `newAdmin` and `removeAdmin` parameters is emitted in the `acceptAdmin()` function as in line 98.

The parameter values of the `NewAdmin` event that is being emitted are incorrect because the same value is used for both parameters. The emitted parameter values should be the `pendingAdmin_`, and the `pendingRemoveAdmin_` values.

BigOwner.sol

```
69  event NewAdmin(address indexed newAdmin, address indexed removeAdmin);
70  function acceptAdmin(address pendingAdmin_, address pendingRemoveAdmin_) public
    {
71      bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
72      require(txHash == pendingAdminHash, 'BigOwner::acceptAdmin: Argument not
    match pendingAdminHash.');
73      require(msg.sender == pendingAdmin_, 'BigOwner::acceptAdmin: Call must come
    from newAdmin.');
74      require(
75          pendingAdminHashSubmitBlock + PANDING_BLOCK >= getBlockNumber(),
76          "BigOwner::acceptAdmin: PendingAdmin hasn't surpassed pending time."
77      );
78
79      admin[pendingAdmin_] = true;
80      admin[pendingRemoveAdmin_] = false;
81
82      uint256 removeAdminIndex = 0;
83      bool isFound = false;
84      for (uint256 i = 0; i < adminList.length; i++) {
85          if (adminList[i] == pendingRemoveAdmin_) {
86              removeAdminIndex = i;
87              isFound = true;
88              break;
89          }
90      }
91      assert(isFound);
92
93      adminList[removeAdminIndex] = pendingAdmin_;
94
95      pendingAdminHashSubmitter = address(0);
96      pendingAdminHash = '';
97
98      emit NewAdmin(pendingAdmin_, pendingAdmin_);
99  }
```

5.6.2. Remediation

Inspex suggests changing the pendingRemoveAdmin_ parameter value from pendingAdmin_ to pendingRemoveAdmin_ variable in line 98 as shown below:

BigOwner.sol

```
69  event NewAdmin(address indexed newAdmin, address indexed removeAdmin);
70  function acceptAdmin(address pendingAdmin_, address pendingRemoveAdmin_) public
    {
71      bytes32 txHash = keccak256(abi.encode(pendingAdmin_, pendingRemoveAdmin_));
72      require(txHash == pendingAdminHash, 'BigOwner::acceptAdmin: Argument not
    match pendingAdminHash.');
73      require(msg.sender == pendingAdmin_, 'BigOwner::acceptAdmin: Call must come
    from newAdmin.');
74      require(
75          pendingAdminHashSubmitBlock + PANDING_BLOCK >= getBlockNumber(),
76          "BigOwner::acceptAdmin: PendingAdmin hasn't surpassed pending time."
77      );
78
79      admin[pendingAdmin_] = true;
80      admin[pendingRemoveAdmin_] = false;
81
82      uint256 removeAdminIndex = 0;
83      bool isFound = false;
84      for (uint256 i = 0; i < adminList.length; i++) {
85          if (adminList[i] == pendingRemoveAdmin_) {
86              removeAdminIndex = i;
87              isFound = true;
88              break;
89          }
```

```
    }
    assert(isFound);

    adminList[removeAdminIndex] = pendingAdmin_;

    pendingAdminHashSubmitter = address(0);
    pendingAdminHash = '';

    emit NewAdmin(pendingAdmin_, pendingRemoveAdmin_);
}
```

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: 18-Jan-2022]

[2] "List of Known Bugs — Solidity 0.8.12 documentation" [Online]. Available: https://docs.soliditylang.org/en/latest/bugs.html. [Accessed: 18-Jan-2022]

[3] "Releases · ethereum/solidity" [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 18-Jan-2022]

inspex

CYBERSECURITY PROFESSIONAL SERVICE