Poll

Smart Contract Audit Report
Prepared for Token X

Date Issued:	Jul 16, 2024
Project ID:	AUDIT2024008
Version:	v1.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2024008
Version	v1.0
Client	Token X
Project	Poll
Auditor(s)	Wachirawit Kanpanluk Peeraphut Punsuwan
Author(s)	Wachirawit Kanpanluk Peeraphut Punsuwan
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jul 16, 2024	Full report	Wachirawit Kanpanluk 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 Token X , Inspex team conducted an audit to verify the security posture of the Poll smart contracts on Jul 12, 2024. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of 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 very low, and 2 info-severity issues. With the project team's prompt response, 1 very low and 1 info-severity issues were resolved in the reassessment, while 1 info-severity issues were acknowledged by the team. Therefore, Inspex trusts that TKXPoll smart contracts have sufficient protections to be safe for public use. However, as the source code is currently not publicly available, there is a potential risk that the smart contracts deployed on the blockchain may not be identical to the audited smart contracts. This discrepancy could result in introducing security vulnerabilities or unintended behaviors that were not identified during the audit process. It is of importance to recognize that interacting with an unverified smart contract may lead to the potential loss of funds. In this case, the hash of the deployed smart contract bytecode should be compared with the hash of the audited smart contract bytecode to ensure that the deployed smart contract is identical to the audited smart contract before interacting with them. In the long run, Inspex suggests resolving all issues found in this report.

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

Token X Poll leverages the ERC-20 token standard to enable token-based voting mechanisms. Participants can use specific ERC-20 tokens to cast votes on various proposals within the contract.

Scope Information:

Project Name	Poll
Website	https://tokenx.finance/
Smart Contract Type	Ethereum Smart Contract
Chain	Token X (TKX)
Programming Language	Solidity
Category	Token

Audit Information:

Audit Method	Whitebox
Audit Date	Jul 12, 2024
Reassessment Date	Jul 16, 2024

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 smart contracts with the following bytecodes were audited and reassessed by Inspex in detail:

Initial Audit:

Contract	Bytecode SHA256 Hash
TKXPoll	bd0a371881d599039eee610a89e96b213afe89a5ee73b7c5ca22b6a288cb66eb

Reassessment Audit:

Contract	Bytecode SHA256 Hash
TKXPoll	d3bd7829cf4c4c02e5a316e8e3bfa9b5a058c18509428c136e8cc5698bc95e09

compiler_config.json

```
{
    solidity: {
        version: "0.8.26",
        settings: {
            optimizer: {
                enabled: true,
                    runs: 200
            }
        }
    }
}
```

As the Token X team has decided not to publish the source code to protect their intellectual property, the users should compare the bytecode hashes with the smart contracts deployed before interacting with them to make sure that they are the same with the contracts audited.

2.3. Security Model

2.3.1 Trust Modules

The `TKXPoll` contract has privileged roles with the authority to mutate the critical state variables of the contract. Changes to these state variables significantly impact the contract's functionality. The privileged roles and their corresponding privileged functions are enumerated as follows:

- The owner address can perform the following actions:
  - Add a new proposal.
  - Modify the configurations of proposals.
  - Terminate the voting period of proposals prematurely.
  - Burn burnable tokens after the termination of a proposal.

2.3.2 Trust Assumptions

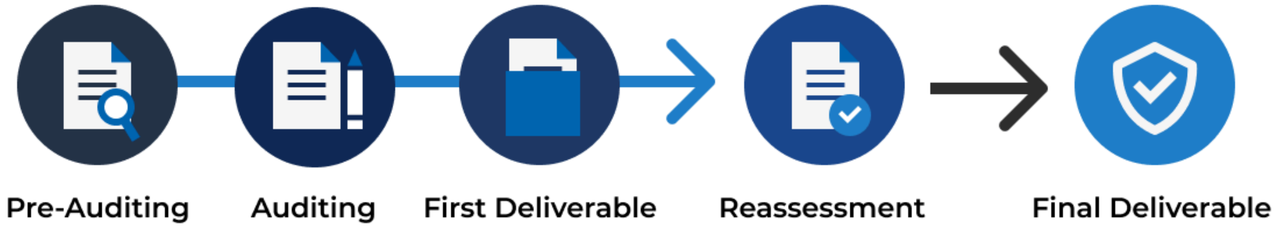
In the `TKXPoll` contract, the protocol's privileged roles, have the ability to change the critical state variables of the contract, also external components were assumed to be trusted. Acknowledging these trust assumptions is important, as it introduces substantial risks to the platform. Trust assumptions include, but are not limited to:

- The owner address is entrusted with the following responsibilities, to be carried out in good faith:
  - Correctly add proposals.
  - Correctly edit the configurations of proposals.
  - Prematurely terminate the voting period of proposals.
  - Withdraw burnable tokens to all users in the event of a force stop of a proposal.

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 testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at (https://docs.inspex.co/smart-contract-security-testing-guide/).

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) 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**.

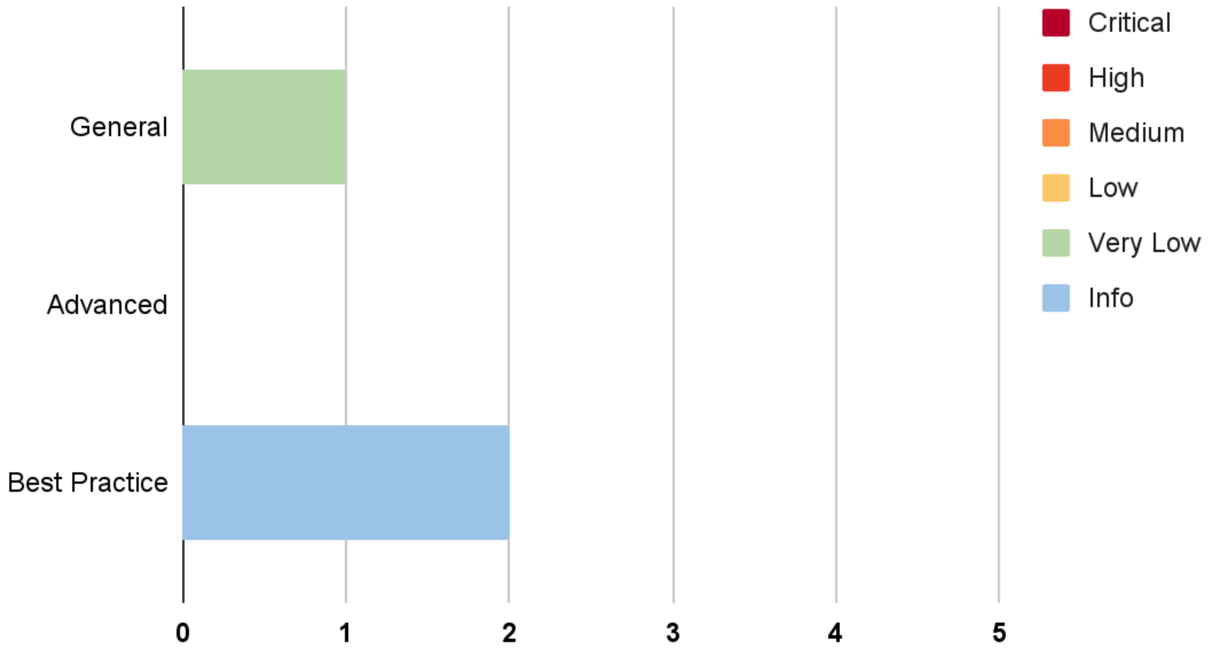
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

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

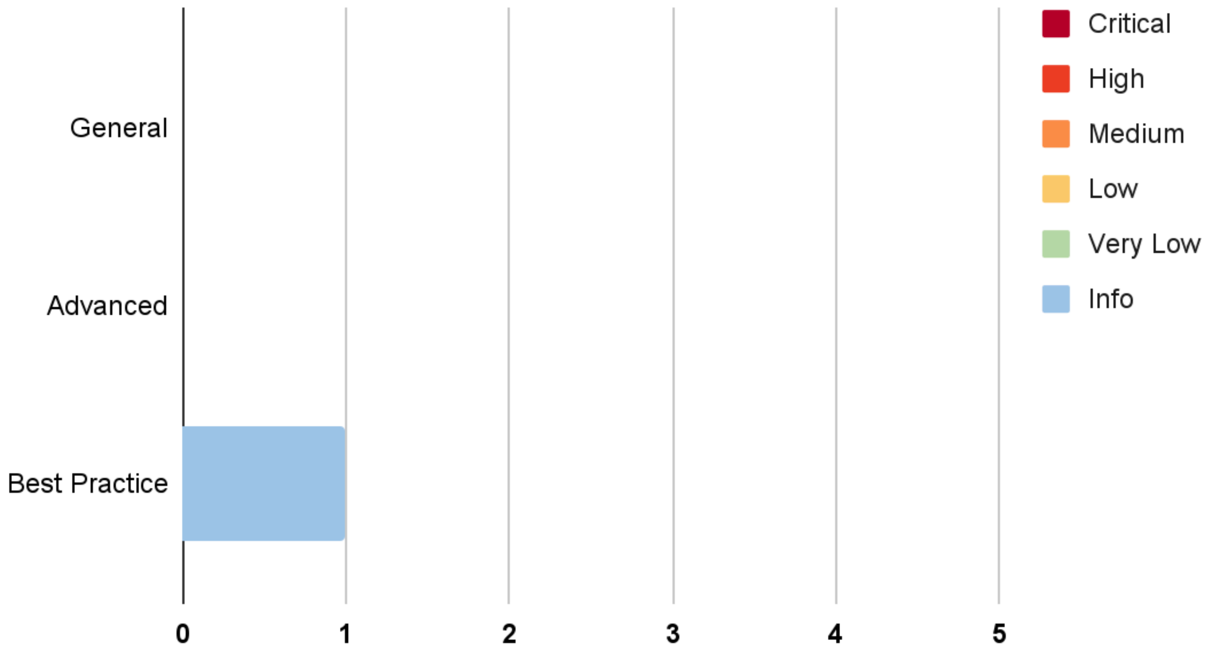
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



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	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-002	Improper Use of Validation	Best Practice	Info	Resolved
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	Info	No Security Impact

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

5. Detailed Findings Information

5.1. Insufficient Logging for Privileged Functions

ID	IDX-001
Target	TKXPoll
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 Token X team has resolved this issue as suggested by adding the emitting of events for the execution of privileged functions.

5.1.1. 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 force stop the proposal from being voted on by executing the `directClosePoll()` function in the `TKXPoll` contract, and no events are emitted.

The privileged functions without sufficient logging are as follows:

File	Contract	Function
TKXPoll.sol (L:169)	TKXPoll	editPollVoteTime()
TKXPoll.sol (L:176)	TKXPoll	directClosePoll()
TKXPoll.sol (L:220)	TKXPoll	editProposalDesc()

5.1.2. Remediation

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

TKXPoll.sol

```
event ClosePoll();

function directClosePoll() external onlyOwner {
    isForceClosed = true;
    emit ClosePoll();
}
```

5.2. Improper Use of Validation

ID	IDX-002
Target	TKXPoll
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Token X team has resolved this issue as suggested by replacing the `assert()` statement with the `require()` statement when the condition is used for input validation or to validate the state before executing certain logic.

5.2.1. Description

The `assert()` statement serves as an assertive check. In a properly functioning smart contract, a failing assert statement should never occur.

Instead, the `require()` statement should be used to validate that the conditions are met or to validate return values from external contract calls.

The following table contains all functions that use the `assert()` statement.

File	Contract	Function
TKXPoll.sol (L:188)	TKXPoll	burnToken()
TKXPoll.sol (L:296)	TKXPoll	_vote()

5.2.2. Remediation

Inspex suggests replacing the `assert()` statement with the `require()` statement if it is used for input validation or to validate the state before executing certain logic, for example in lines 195 and 200:

TKXPoll.sol

```
188 function burnToken() external onlyOwner onlyFinished returns (bool) {
189     require(burnType, 'Poll [burnToken]: requires Poll of burn type.');
190 
191     uint256 lockedBalance = voteToken.balanceOf(address(this));
192     uint256 burnAmount = (lockedBalance * burnPercent) / 100;
193 
194     // Transfer burn amount to burn address
195     require(voteToken.transfer(BURN_ADDRESS, burnAmount));
196 
197     uint256 remain = voteToken.balanceOf(address(this));
198 
199     // Transfer remain to receive wallet
200     require(voteToken.transfer(receiveAddress, remain));
201 
202     emit Burn(address(this), address(voteToken), burnAmount);
203 
204     return true;
205 }
```

5.3. Inexplicit Solidity Compiler Version

ID	IDX-003
Target	TKXPoll
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact The Token X team has acknowledged this issue, determining that accepting the associated risk currently has no direct impact.

5.3.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

TKXPoll.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

5.3.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.26. (https://github.com/ethereum/solidity/releases)

TKXPoll.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
```

For chains that may not be compatible with Solidity compiler version 0.8.26, Inspex suggests using Solidity compiler version 0.8.19 instead, as Solidity compiler version 0.8.20 or later introduces the PUSH0 (0x5f) opcode, which some chains have not yet included. (https://github.com/ethereum/solidity/releases/tag/v0.8.20)

TKXPoll.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;
```

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

inspex

CYBERSECURITY PROFESSIONAL SERVICE