PRESTIGE

Smart Contract Audit Report
Prepared for Token X

Date Issued:	Jun 14, 2022
Project ID:	AUDIT2022038
Version:	v2.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022038
Version	v2.0
Client	Token X
Project	PRESTIGE
Auditor(s)	Patipon Suwanbol Fungkiat Phadejtaku
Author(s)	Darunphop Pengkumta
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	Jun 14, 2022	Update information and confidentiality level	Patipon Suwanbol
1.0	Jun 7, 2022	Full report	Darunphop Pengkumta

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, an ICO portal in Thailand, which is granted the approval by the Securities and Exchange Commission (SEC), Thailand, Inspex team conducted an audit to verify the security posture of the PRESTIGE smart contracts on Jun 1, 2022. The smart contract code covered in this assignment will be used by Token X on its platform in the future for the purpose of investment token offerings for its potential customers and issuers thereafter.

During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of PRESTIGE 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

Based on the audit performed, nothing has come to our attention that would cause us to believe the smart contract code in the PRESTIGE project is not faithfully implemented in all material respects in accordance with Token X specification. During the audit, Inspex found 1 medium and 2 info-severity issues. With the project team's prompt response, 2 info-severity issues were resolved in the reassessment, while 1 medium-severity issue was mitigated by the team. Therefore, Inspex trusts that PRESTIGE smart contracts have high-level protections in place to be safe from most attacks.

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, Inspex is independent of the client, and nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

PRESTIGE is an investment token with additional features: pause and burn. This token is allowed to be used by a specific group of users based on a list in the AllowlistRegistry contract (whitelist mechanism). However, the smart contract can include digital asset exchanges in the whitelist, allowing the token to be traded by the users in the exchanges. In addition, the contract provides the emergency functions for the authorized party to handle incidents promptly complying with the regulator's guidelines.

Scope Information:

Project Name	PRESTIGE
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	Jun 1, 2022
Reassessment Date	Jun 6, 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: eb7733c44d3bb790f76dadef929910f6bad30618)

Contract	Location (URL)
AllowlistRegistry	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/eb7733c44d/contracts/AllowlistRegistry.sol
InvestmentToken	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/eb7733c44d/contracts/InvestmentToken.sol
EmergencyWithdrawable	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/eb7733c44d/extensions/EmergencyWithdrawable.sol
ERC20AllowListableProxy	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/eb7733c44d/extensions/ERC20AllowListableProxy.sol

Reassessment: (Commit: a9d84e3a53e2e6c365ab2b1f3d517ec735626bb4)

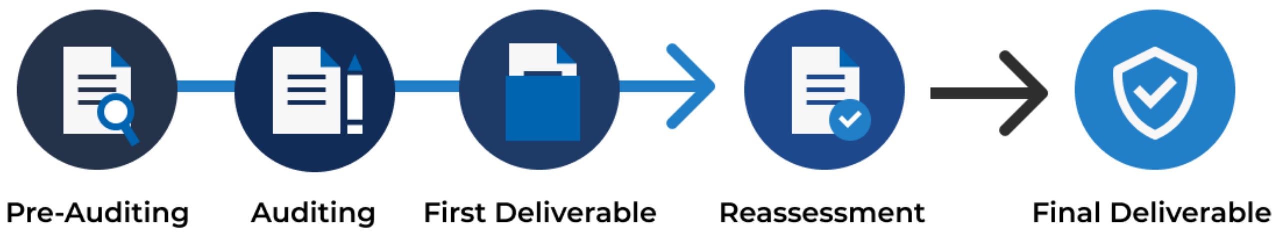
Contract	Location (URL)
AllowlistRegistry	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/a9d84e3a53/contracts/AllowlistRegistry.sol
InvestmentToken	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/a9d84e3a53/contracts/InvestmentToken.sol
EmergencyWithdrawable	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/a9d84e3a53/extensions/EmergencyWithdrawable.sol
ERC20AllowListableProxy	https://github.com/tokenx-finance/park-luxury-prestige-smart-contracts/blob/a9d84e3a53/extensions/ERC20AllowListableProxy.sol

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 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://inspex.gitbook.io/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**.

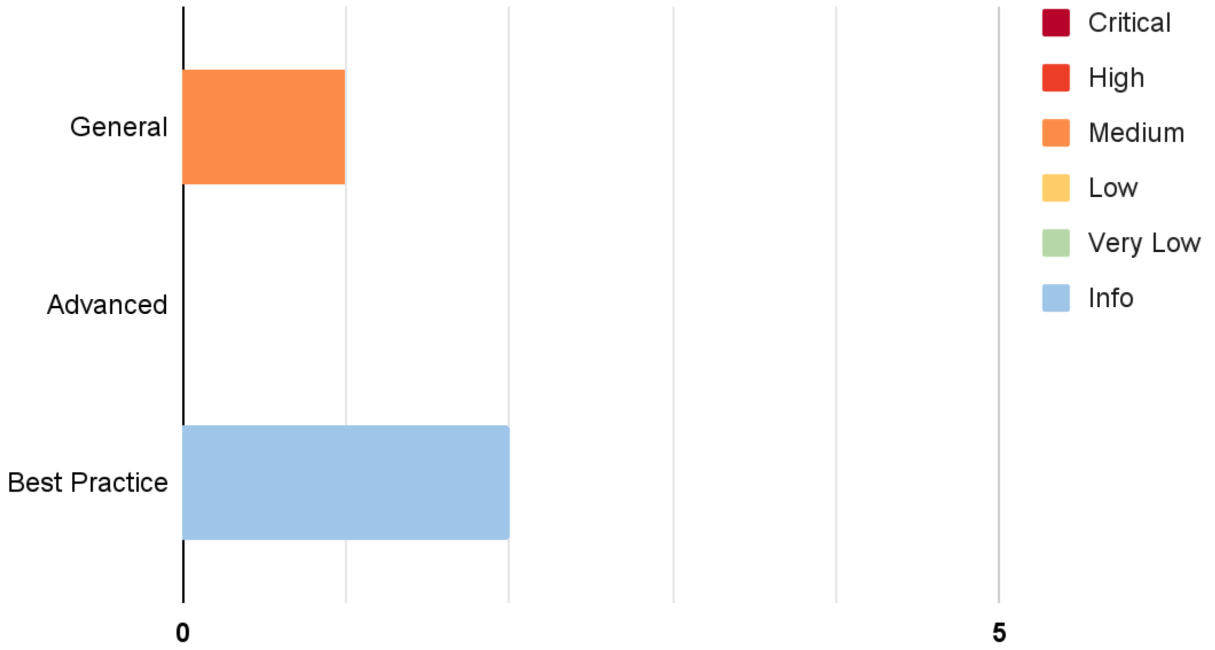
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would 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	Centralized Control of State Variable	General	Medium	Resolved *
IDX-002	Improper Function Visibility	Best Practice	Info	Resolved
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved

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

5. Detailed Findings Information

5.1. Centralized Control of State Variable

ID	IDX-001
Target	AllowlistRegistry InvestmentToken
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** High The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Low There is nothing to restrict the changes from being done; however, this action can only be done by the authorized party which is a multisig wallet.
Status	Resolved * The centralized functions are required to comply with the regulator's guidelines for protecting users' funds, also known as clawback mechanism. As a result, the Token X team has mitigated this issue by delegating control to a multi-sig wallet for those privileged functions to ensure that all the critical actions are properly reviewed according to the regulation. Those functions will be controlled by multi-sig parties, including the issuer of the token and Token X, requiring both parties to agree on executing those functions.

5.1.1. Description

Critical state variables can be updated at any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users. For example, the platform owner can call the `setAllowlistRegistry()` function to change the `registry` address and then the existing whitelisted users will become non-whitelisted users, resulting in being unable to execute the `InvestmentToken` contract's functionalities for those users.

The functions that can be used to update critical state variables are as follows:

File	Contract	Function	Modifier
AllowlistRegistry.sol (L:40)	AllowlistRegistry	addAllowlist()	onlyOwner

AllowlistRegistry.sol (L:53)	AllowlistRegistry	removeAllowlist()	onlyOwner
InvestmentToken.sol (L:49)	InvestmentToken	setAllowlistRegistry()	onlyOwner
InvestmentToken.sol (L:186)	InvestmentToken	adminTransfer()	onlyOwner
InvestmentToken.sol (L:199)	InvestmentToken	adminBurn()	onlyOwner
InvestmentToken.sol (L:212)	InvestmentToken	pause()	onlyOwner
InvestmentToken.sol (L:225)	InvestmentToken	unpause()	onlyOwner

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time, e.g., 24 hours. However, if the timelock is used as a mitigation, the `onlyOwner` modifier should be changed to another operation role such as `onlyOperator` to prevent the timelock effect from applying to other functions with the `onlyOwner` modifier.

In addition, a multi-sig wallet, a wallet that requires a majority of votes from the controlling parties to be passed before allowing the execution of a transaction, can be used as a mitigation to ensure that all privilege actions are well prepared.

5.2. Improper Function Visibility

ID	IDX-002
Target	AllowlistRegistry InvestmentToken ERC20AllowListableProxy
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 by changing the visibility of the suggested functions to external in commit `a9d84e3a53e2e6c365ab2b1f3d517ec735626bb4`.

5.2.1. Description

Functions which apply `public` visibility without being called internally by the functions in the contract itself should have external visibility. This improves the readability of the contract, allowing clear distinction between functions that are externally used and functions that are also called internally.

For example, the following source code shows that the `isAllowlist()` function in the `AllowlistRegistry` contact is set to public, but it is never called from any internal functions.

AllowlistRegistry.sol

```
26 /**
27 * @dev Returns the allowlist status of an account.
28 */
29 function isAllowlist(address account) public view virtual returns (bool) {
30     return _allowlist[account];
31 }
```

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

Target	Contract	Function
AllowlistRegistry.sol (L:29)	AllowlistRegistry	isAllowlist()
AllowlistRegistry.sol (L:40)	AllowlistRegistry	addAllowlist()

AllowlistRegistry.sol (L:53)	AllowlistRegistry	removeAllowlist()
ERC20AllowListableProxy.sol (L:21)	ERC20AllowListableProxy	allowlistRegistry()
InvestmentToken.sol (L:49)	InvestmentToken	setAllowlistRegistry()
InvestmentToken.sol (L:186)	InvestmentToken	adminTransfer()
InvestmentToken.sol (L:199)	InvestmentToken	adminBurn()
InvestmentToken.sol (L:212)	InvestmentToken	pause()
InvestmentToken.sol (L:225)	InvestmentToken	unpause()
InvestmentToken.sol (L:238)	InvestmentToken	emergencyWithdrawToken()

5.2.2. Remediation

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

AllowlistRegistry.sol

```
/**
 * @dev Returns the allowlist status of an account.
 */
function isAllowlist(address account) external view virtual returns (bool) {
    return _allowlist[account];
}
```

5.3. Inexplicit Solidity Compiler Version

ID	IDX-003
Target	AllowlistRegistry InvestmentToken EmergencyWithdrawable ERC20AllowListableProxy
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Token X team has resolved this issue by changing the compiler version in the contracts into the newest version in commit `a9d84e3a53e2e6c365ab2b1f3d517ec735626bb4`.

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.

AllowlistRegistry.sol

```
// SPDX-License-Identifier: Unlicense
// TokenX Contracts v1.0.0 (extensions/AllowlistRegistry.sol)
pragma solidity ^0.8.0;
```

The following table contains all contracts which the inexplicit compiler is declared.

Contract	Version
AllowlistRegistry	^0.8.0
InvestmentToken	^0.8.0
EmergencyWithdrawable	^0.8.0
ERC20AllowListableProxy	^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 v0.8.14 (https://github.com/ethereum/solidity/releases/tag/v0.8.14).

AllowlistRegistry.sol

```
// SPDX-License-Identifier: Unlicense
// TokenX Contracts v1.0.0 (extensions/AllowlistRegistry.sol)
pragma solidity 0.8.14;
```

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