ICOToken, TokenLocker & TokenSale

Smart Contract Audit Report
Prepared for Cunicorn

Date Issued:	Jun 8, 2022
Project ID:	AUDIT2022015
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022015
Version	v1.0
Client	Cunicorn
Project	ICOToken, TokenLocker & TokenSale
Auditor(s)	Patipon Suwanbol Ronnachai chaipha
Author(s)	Wachirawit Kanpanluk
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jun 8, 2022	Full report	Wachirawit Kanpanluk

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 Cunicorn, Inspex team conducted an audit to verify the security posture of the ICOToken, TokenLocker & TokenSale smart contracts between Mar 1, 2022 and Mar 2, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of ICOToken, TokenLocker & TokenSale 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 high, 1 low, 1 very low, and 2 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that ICOToken, TokenLocker & TokenSale 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, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Cunicorn provides digital transformation and blockchain-based solutions for the users. The protocol has been developed by a group of tech developers, product specialists, and financial experts, who are passionate about the technology of tomorrow and have no fear of running through brick walls.

An ICOToken contract is a governance token that is used as the medium of exchange on the platform. The TokenSale contract allows users to buy any token that is offered by the platform in terms of an "Initial Coin Offering" (ICO), in which these purchased tokens will be able to be redeemed in the TokenLocker contract.

Scope Information:

Project Name	ICOToken, TokenLocker & TokenSale
Website	https://cunicorn.co
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	Token

Audit Information:

Audit Method	Whitebox
Audit Date	Mar 1, 2022 - Mar 2, 2022
Reassessment Date	May 18, 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: 9965caabefddbcf6fd0d3207170035f2b48a5e23)

Contract	Location (URL)
ICOToken	https://github.com/cunicorn-official/tokensale-contract/blob/9965caabef/contracts/ICOToken.sol
TokenSale	https://github.com/cunicorn-official/tokensale-contract/blob/9965caabef/contracts/TokenSale.sol
TokenLocker	https://github.com/cunicorn-official/tokensale-contract/blob/9965caabef/contracts/TokenLocker.sol

Reassessment: (Commit: e9fd2e3e7d372496987ce2d34a5de9878715fdd5)

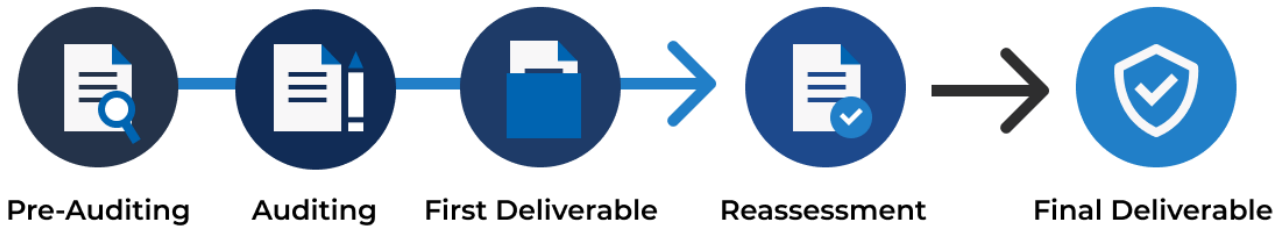
Contract	Location (URL)
ICOToken	https://github.com/cunicorn-official/tokensale-contract/blob/e9fd2e3e7d/contracts/ICOToken.sol
TokenSale	https://github.com/cunicorn-official/tokensale-contract/blob/e9fd2e3e7d/contracts/TokenSale.sol
TokenLocker	https://github.com/cunicorn-official/tokensale-contract/blob/e9fd2e3e7d/contracts/TokenLocker.sol

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**.

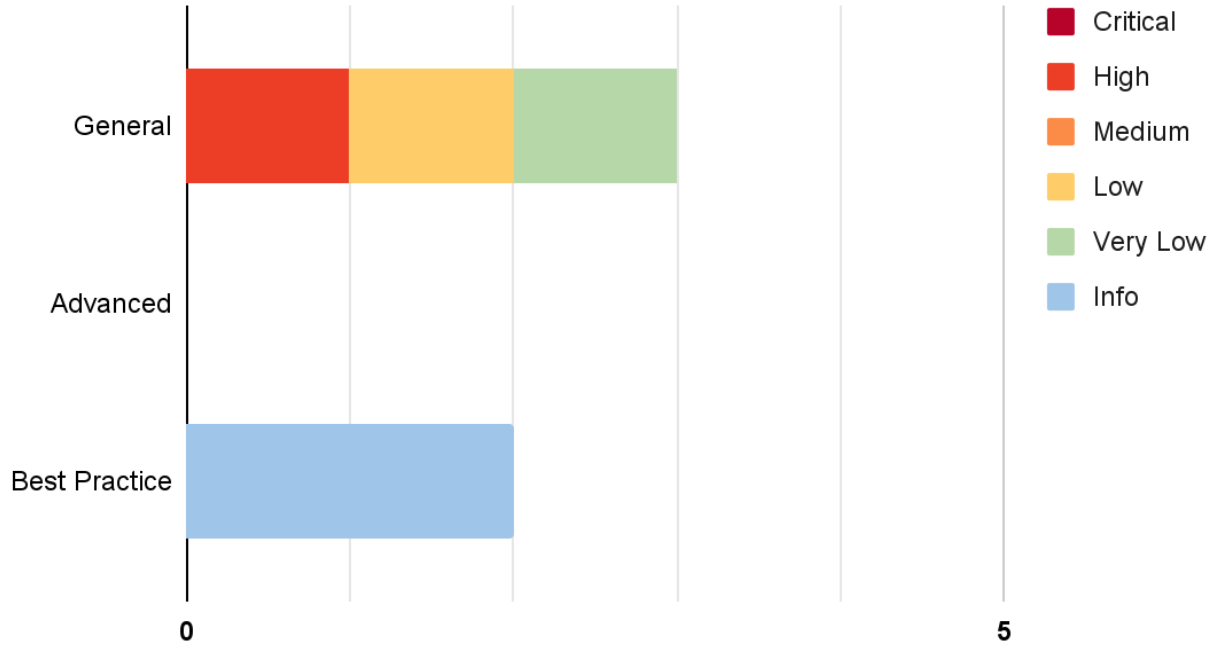
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

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

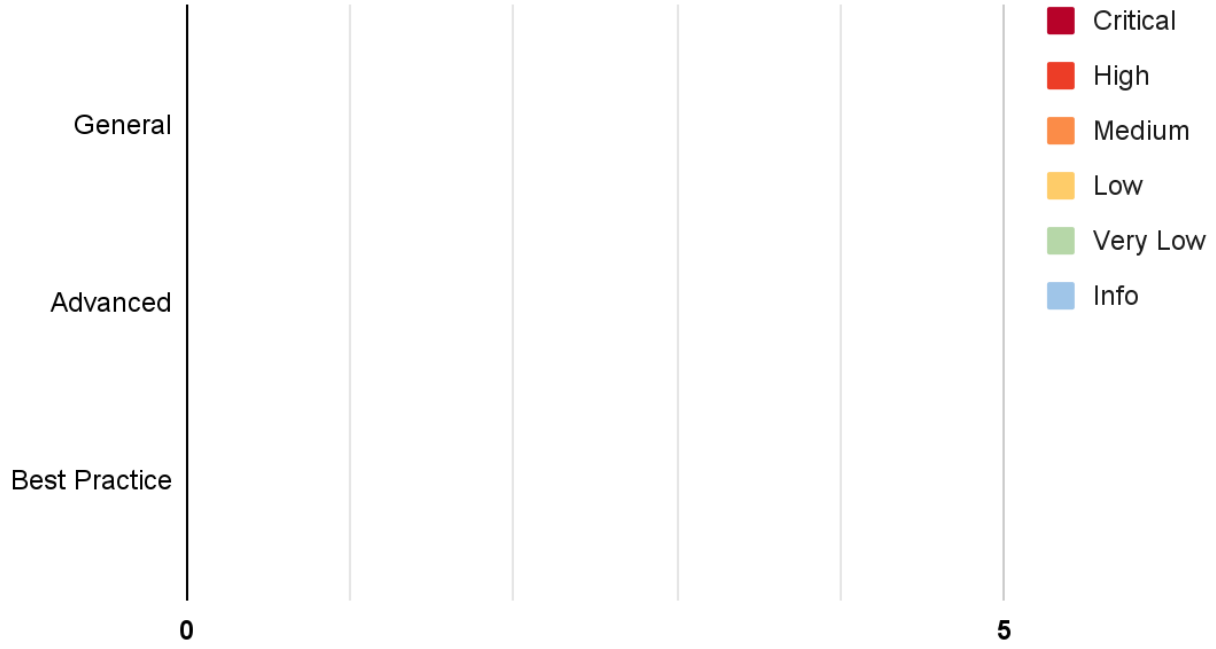
4. Summary of Findings

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	Use of Upgradable Contract	General	High	Resolved *
IDX-002	Centralized Control of State Variable	General	Low	Resolved *
IDX-003	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-004	Improper Function Visibility	Best Practice	Info	Resolved
IDX-005	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved

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

5. Detailed Findings Information

5.1. Use of Upgradable Contract

ID	IDX-001
Target	ICOToken TokenSale TokenLock
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The logic of the affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the users' funds anytime they want. **Likelihood: Medium** This action can be performed by the proxy owner without any restriction.
Status	Resolved * Cunicorn team has confirmed that the upgrade mechanism will be called through the `Timelock` contract. This allows the platform users to monitor the timelock and be notified of the potential changes being made to the smart contracts. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the `Timelock` contract before using them.

5.1.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As these smart contracts are upgradeable, the logic of them can be modified by the owner anytime, making the smart contracts untrustworthy.

5.1.2. Remediation

Inspex suggests deploying the contracts without the proxy pattern or any solution that can make smart contracts upgradeable.

However, if upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes e.g., 1 day. This allows the platform users to monitor the timelock and is notified of the potential changes being done on the smart contracts.

5.2. Centralized Control of State Variable

ID	IDX-002
Target	TokenSale
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Low The controlling authorities can change the critical state variables, thereby causing unfairness to the other users. However, the impact is limited to the user's opportunities for buying tokens only, which does not affect the monetary loss for the users. **Likelihood:** Medium There is nothing to restrict the changes from being done by the owner. However, only some owner roles can call these functions to change the states.
Status	Resolved * Cunicorn team has confirmed that the privilege function will be called through the `Timelock` contract. This means any action that would occur to the privilege function will be able to be monitored by the community conveniently. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the `Timelock` contract before using them.

5.2.1. 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.

The controllable privileged state update functions are as follows:

File	Contract	Function	Role
TokenSale.sol (L:202)	TokenSale	setSalePeriod()	CONFIG_ROLE
TokenSale.sol (L:215)	TokenSale	setWhitelistPeriod()	CONFIG_ROLE
TokenSale.sol (L:237)	TokenSale	setLimitTokenReceivePerUser()	CONFIG_ROLE

5.2.2. 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, as the `addWhitelistUser()` function also applies the role of `CONFIG_ROLE`, if Cunicorn team decides to mitigate this issue by using the timelock mechanism, to avoid delays, it is suggested to change the required role for executing the `addWhitelistUser()` function to others.

5.3. Insufficient Logging for Privileged Functions

ID	IDX-003
Target	TokenSale
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 Cunicorn team has resolved this issue as suggested by emitting events for the execution of privileged functions in commit `a5bff682b744036326eb78a526be2e0994798d8c`.

5.3.1. 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 set the sale period by executing the `setSalePeriod()` function in the `TokenSale` contract, and no events are emitted.

TokenSale.sol

```
202 function setSalePeriod(uint48 _tokenSaleStartTime, uint48 _tokenSaleEndTime)
    external onlyRole(CONFIG_ROLE) {
203     require(_tokenSaleStartTime != 0, "TOKENSALE: INPUT_ZERO_AMOUNT");
204     require(_tokenSaleEndTime != 0, "TOKENSALE: INPUT_ZERO_AMOUNT");
205     require(_tokenSaleStartTime < _tokenSaleEndTime, "TOKENSALE:
    CONFIG_INVALID_SALE_TIME");
206     // set sale period
207     tokenSaleStartTime = _tokenSaleStartTime;
208     tokenSaleEndTime = _tokenSaleEndTime;
209 }
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function
TokenSale.sol (L:192)	TokenSale	addWhitelistUser()
TokenSale.sol (L:202)	TokenSale	setSalePeriod()
TokenSale.sol (L:215)	TokenSale	setWhitelistPeriod()
TokenSale.sol (L:237)	TokenSale	setLimitTokenReceivePerUser()
TokenSale.sol (L:247)	TokenSale	setPauseSale()

5.3.2. Remediation

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

TokenSale.sol

```
201 event SetSalePeriod(uint48 tokenSaleStartTime, uint48 tokenSaleEndTime);
202 function setSalePeriod(uint48 _tokenSaleStartTime, uint48 _tokenSaleEndTime)
    external onlyRole(CONFIG_ROLE) {
203     require(_tokenSaleStartTime != 0, "TOKENSALE: INPUT_ZERO_AMOUNT");
204     require(_tokenSaleEndTime != 0, "TOKENSALE: INPUT_ZERO_AMOUNT");
205     require(_tokenSaleStartTime < _tokenSaleEndTime, "TOKENSALE:
    CONFIG_INVALID_SALE_TIME");
206     // set sale period
207     tokenSaleStartTime = _tokenSaleStartTime;
208     tokenSaleEndTime = _tokenSaleEndTime;
209     emit SetSalePeriod(_tokenSaleStartTime, _tokenSaleEndTime);
210 }
```

5.4. Improper Function Visibility

ID	IDX-004
Target	ICOToken TokenSale TokenLocker
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Cunicorn team has resolved this issue as suggested by changing the functions' visibility to `external` in commit `a5bff682b744036326eb78a526be2e0994798d8c`.

5.4.1. Description

Public functions that are never called internally by 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.

The following source code shows that the `initialize()` function of the `ICOToken` contract is set to public and it is never called from any internal function.

ICOToken.sol

```
17  function initialize(string memory name_, string memory symbol_, uint256 cap_)
    public initializer {
18      require(cap_ > 0, "ERC20Capped: cap is 0");
19
20      __ERC20_init(name_, symbol_);
21      __ERC20Burnable_init();
22      __Pausable_init();
23      __AccessControl_init();
24      __ERC20Permit_init(name_);
25
26      _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
27      _setupRole(PAUSER_ROLE, msg.sender);
28      _setupRole(MINTER_ROLE, msg.sender);
29
30      _cap = cap_;
31  }
```

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

File	Contract	Function
ICOToken.sol (L:17)	ICOToken	initialize()
ICOToken.sol (L:33)	ICOToken	pause()
ICOToken.sol (L:37)	ICOToken	unpause()
ICOToken.sol (L:45)	ICOToken	mint()
TokenSale.sol (L:129)	TokenSale	initialize()
TokenLocker.sol (L:82)	TokenLocker	initialize()

5.4.2. Remediation

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

ICOToken.sol

```
17  function initialize(string memory name_, string memory symbol_, uint256 cap_)
    external initializer {
18      require(cap_ > 0, "ERC20Capped: cap is 0");
19
20      __ERC20_init(name_, symbol_);
21      __ERC20Burnable_init();
22      __Pausable_init();
23      __AccessControl_init();
24      __ERC20Permit_init(name_);
25
26      _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
27      _setupRole(PAUSER_ROLE, msg.sender);
28      _setupRole(MINTER_ROLE, msg.sender);
29
30      _cap = cap_;
31  }
```

5.5. Inexplicit Solidity Compiler Version

ID	IDX-005
Target	ICOToken TokenSale TokenLocker
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Cunicorn team has resolved this issue as suggested by fixing the Solidity compiler to the latest stable version in commit `a5bff682b744036326eb78a526be2e0994798d8c`.

5.5.1. 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.

TokenSale.sol

```
1 // SPDX-License-Identifier: BUSL-1.1
2 pragma solidity ^0.8.0;
```

The following table contains all targets which have the inexplicit solidity compiler version.

File	Contract	Version
ICOToken.sol (L:2)	TokenSale	^0.8.2
TokenSale.sol (L:2)	TokenSale	^0.8.0
TokenLocker.sol (L:2)	TokenSale	^0.8.0

5.5.2. 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 v0.8.13 (https://github.com/ethereum/solidity/releases/tag/v0.8.13).

TokenSale.sol

```
1 // SPDX-License-Identifier: BUSL-1.1
2 pragma solidity 0.8.13;
```

6. Appendix

6.1. About Inspex

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