

# Coin98VRC25

## Smart Contract Audit Report Prepared for Coin98



---

**Date Issued:** Sep 5, 2023  
**Project ID:** AUDIT2023017  
**Version:** v1.0  
**Confidentiality Level:** Public

## Report Information

Project ID	AUDIT2023017
Version	v1.0
Client	Coin98
Project	Coin98VRC25
Auditor(s)	Wachirawit Kanpanluk Sorawish Laovakul
Author(s)	Wachirawit Kanpanluk Sorawish Laovakul
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

## Version History

Version	Date	Description	Author(s)
1.0	Sep 5, 2023	Full report	Wachirawit Kanpanluk Sorawish Laovakul

## Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	<a href="https://t.me/inspexco">t.me/inspexco</a>
Email	<a href="mailto:audit@inspex.co">audit@inspex.co</a>

---

## Table of Contents

<b>1. Executive Summary</b>	<b>1</b>
1.1. Audit Result	1
1.2. Disclaimer	1
<b>2. Project Overview</b>	<b>2</b>
2.1. Project Introduction	2
2.2. Scope	3
<b>3. Methodology</b>	<b>4</b>
3.1. Test Categories	4
3.2. Audit Items	5
3.3. Risk Rating	7
<b>4. Summary of Findings</b>	<b>8</b>
<b>5. Detailed Findings Information</b>	<b>10</b>
5.1. Centralized Control of State Variable	10
5.2. Unusable Token via Contract Freezing	12
5.3. Inexplicit Solidity Compiler Version	14
5.4. Improper Function Visibility	15
<b>6. Appendix</b>	<b>17</b>
6.1. About Inspect	17

## 1. Executive Summary

As requested by Coin98, Inspex team conducted an audit to verify the security posture of the Coin98VRC25 smart contracts on Aug 29, 2023. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Coin98VRC25 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 medium, 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 Coin98VRC25 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

The Coin98VRC25 token functions as the utility token for the Coin98 platform. It will serve purposes such as covering service fees, acting as staking incentives, facilitating governance, and providing exclusive membership rights. Token holders can also conduct transfers while using the token to pay gas fees instead of the native Tomochain network token. An alternative application includes enabling gas-sponsored smart contracts.

#### Scope Information:

Project Name	Coin98VRC25
Website	<a href="https://coin98.com/">https://coin98.com/</a>
Smart Contract Type	Ethereum Smart Contract
Chain	Tomochain
Programming Language	Solidity
Category	Token

#### Audit Information:

Audit Method	Whitebox
Audit Date	Aug 29, 2023
Reassessment Date	Sep 5, 2023

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 Inspect in detail:

**Initial Audit: (Commit: 99506cbc9917f19d4db225434034db44c4909e3)**

Contract	Location (URL)
Coin98VRC25	<a href="https://github.com/coin98/coin98-token-contract/blob/99506cbc9/contracts/vrc25/Coin98Token.sol">https://github.com/coin98/coin98-token-contract/blob/99506cbc9/contracts/vrc25/Coin98Token.sol</a>

**Reassessment: (Commit: 6f35a60f0d842763da88f09ec8a28822a9b5b77f)**

Contract	Location (URL)
Coin98VRC25	<a href="https://github.com/coin98/coin98-token-contract/blob/6f35a60f0d/contracts/vrc25/Coin98Token.sol">https://github.com/coin98/coin98-token-contract/blob/6f35a60f0d/contracts/vrc25/Coin98Token.sol</a>
TimelockStandalone	<a href="https://github.com/coin98/coin98-token-contract/blob/6f35a60f0d/contracts/TimelockStandalone.sol">https://github.com/coin98/coin98-token-contract/blob/6f35a60f0d/contracts/TimelockStandalone.sol</a>

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](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](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**.

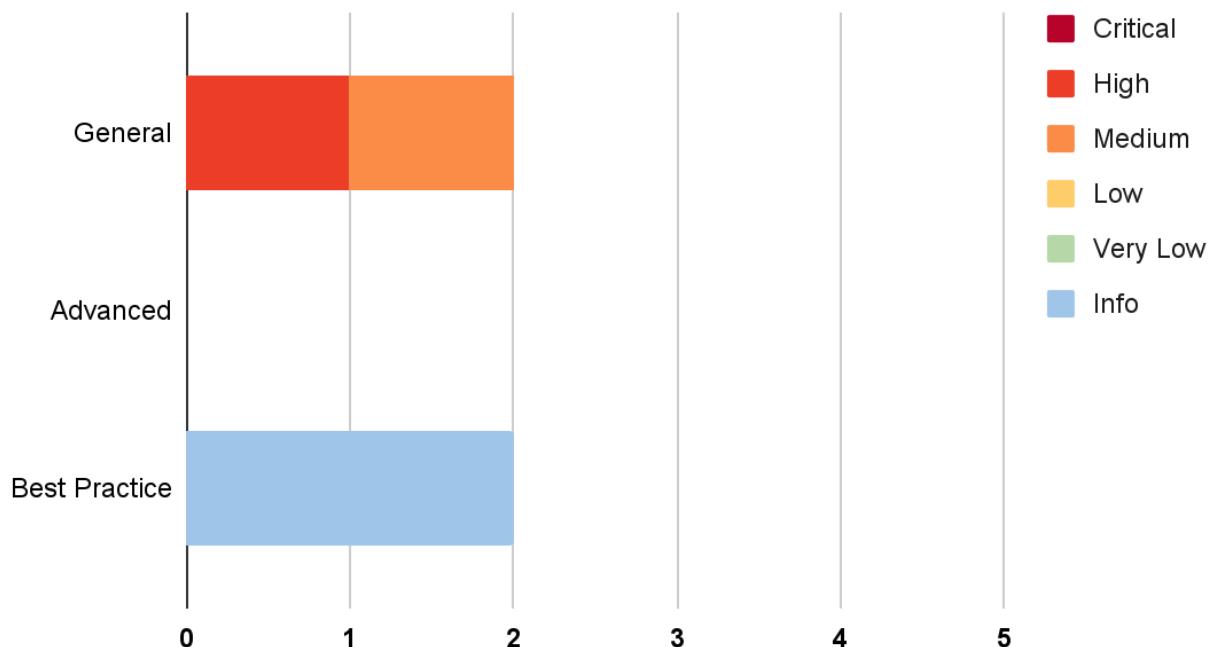
**Severity** is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

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

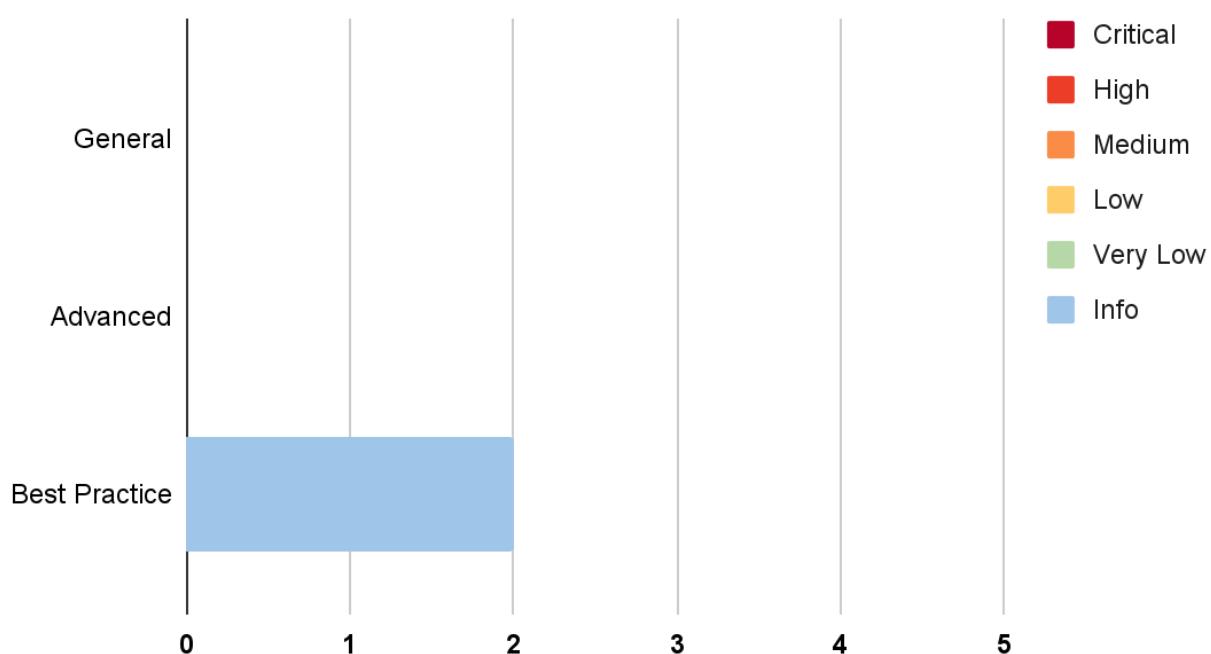
## 4. Summary of Findings

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	Centralized Control of State Variable	General	High	Resolved *
IDX-002	Unusable Token via Contract Freezing	General	Medium	Resolved *
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	Info	No security Impact
IDX-004	Improper Function Visibility	Best Practice	Info	No security Impact

\* The mitigations or clarifications by Coin98 can be found in Chapter 5.

## 5. Detailed Findings Information

### 5.1. Centralized Control of State Variable

ID	IDX-001
Target	Coin98VRC25
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	<p><b>Severity: High</b></p> <p><b>Impact: High</b> The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users.</p> <p><b>Likelihood: Medium</b> There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.</p>
Status	<p><b>Resolved *</b></p> <p>The Coin98 team has mitigated this issue by implementing a timelock contract as the owner of the contract to prevent immediate changes to the contract in commit <a href="#">6f35a60f0d842763da88f09ec8a28822a9b5b77f</a>.</p>

#### 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.

The controllable privileged state update functions are as follows:

Target	Function	Modifier
Coin98Token.sol(L:911)	mint()	onlyOwner
Coin98Token.sol(L:971)	setFee()	onlyOwner
Coin98Token.sol(L:1192)	transferOwnership()	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 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 at least 24 hours.

## 5.2. Unusable Token via Contract Freezing

ID	IDX-002
Target	Coin98VRC25
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	<p><b>Severity:</b> Medium</p> <p><b>Impact:</b> High</p> <p>The holders cannot use tokens freely, resulting in loss of opportunities to gain profit from using the token.</p> <p><b>Likelihood:</b> Low</p> <p>It is very unlikely that the contract owner will freeze the tokens in the normal situation because there is no profit for the owner when the token contract is frozen.</p>
Status	<p><b>Resolved *</b></p> <p>The Coin98 team has mitigated this issue by implementing a timelock mechanism for <code>freeze()</code> and <code>unfreeze()</code> functions in commit  <code>6f35a60f0d842763da88f09ec8a28822a9b5b77f</code>.</p>

### 5.2.1. Description

In the `Coin98VRC25` contract, the owner has the ability to invoke the `freeze()` function, which sets the `_frozen` state to `true`.

#### `Coin98Token.sol`

```

954  function freeze() public onlyOwner {
955      _freeze();
956 }
```

```

1253 function _freeze() internal whenNotFrozen {
1254     _frozen = true;
1255     emit Frozen(_msgSender());
1256 }
```

As a result, the transfer process will consistently fail due to the `_beforeTokenTransfer()` function.

#### Coin98Token.sol

```
1135 function _beforeTokenTransfer(address from, address to, uint256 amount)
1136     internal {
1137         require(!frozen(), "ERC20: token transfer while frozen");
```

This leads to the prevention of all token holders from utilizing their tokens when the contract owner chooses to freeze it.

#### 5.2.2. Remediation

Inspex suggests removing the freeze mechanism and removing the validation check in the `_beforeTokenTransfer()` function.

#### Coin98Token.sol

```
1135 function _beforeTokenTransfer(address from, address to, uint256 amount)
1136     internal {
1137         /* No validation check when frozen */
```

## 5.3. Inexplicit Solidity Compiler Version

ID	IDX-003
Target	Coin98VRC25
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>No security Impact</b> The Coin98 team has acknowledged this issue. However, the inexplicit compiler version 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.

#### Coin98Token.sol

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity >=0.6.0 <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 0.7 is v0.7.6.

#### Coin98Token.sol

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.7.6;

```

## 5.4. Improper Function Visibility

ID	IDX-004
Target	Coin98VRC25
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>No security Impact</b> The Coin98 team has acknowledged this issue. However, improper function visibility has no security impact

### 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 `mint()` function of the `Coin98VRC25` contract is set to public and it is never called from any internal function.

#### Coin98Token.sol

```

911 function mint(address account, uint256 amount) public onlyOwner {
912     _mint(account, amount);
913 }
```

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

Target	Function
Coin98Token.sol(L:911)	mint()
Coin98Token.sol(L:920)	burn()
Coin98Token.sol(L:937)	burnFrom()
Coin98Token.sol(L:954)	freeze()
Coin98Token.sol(L:964)	unfreeze()
Coin98Token.sol(L:971)	setFee()
Coin98Token.sol(L:1179)	acceptOwnership()
Coin98Token.sol(L:1192)	transferOwnership()

#### 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:

##### Coin98Token.sol

```

911 function mint(address account, uint256 amount) external onlyOwner {
912     _mint(account, amount);
913 }
```

## 6. Appendix

### 6.1. About Inspect



## 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	<a href="https://inspex.co">https://inspex.co</a>
Twitter	<a href="https://twitter.com/InspexCo">@InspexCo</a>
Facebook	<a href="https://www.facebook.com/InspexCo">https://www.facebook.com/InspexCo</a>
Telegram	<a href="https://t.me/inspex_announcement">@inspex_announcement</a>



**inspex**

CYBERSECURITY PROFESSIONAL SERVICE