ExVul

SMART CONTRACT AUDIT REPORT

tokimonsterAI Smart Contract

MAY 2025

Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **tokimonsterAI** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood	*Informational*	*Low*	*Medium*	*High*
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

IMPACT

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on

our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
	Abnormal Resource Consumption

Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

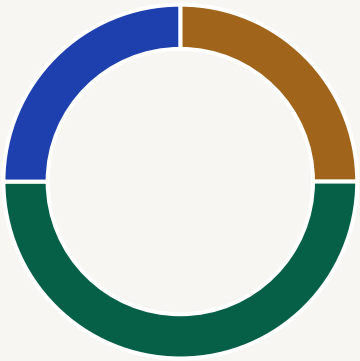
2.1 Project Info And Contract Address

Project Name	Audit Time	Language
tokimonsterAI	May 14 2025 - May 20 2025	Move

Source code	Link
tokimonsterAI	a0da5993e9e5cec754d4c8b1b722dd6b7e319f2a

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	2
INFO	1



2.3 Key Findings

Severity	Findings Title	Status
MEDIUM	Excessive Administrator Privileges in update_team_reward Function	Acknowledge
LOW	Bypassing team_reward Limitation	Fixed
LOW	Invalid Address Replacement in replace_user_reward_recipient Function	Fixed
INFO	Spelling Errors and Naming Conventions	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Excessive Administrator Privileges in update_team_reward Function

Location	Severity	Category
TokimonsterRewarder.move	MEDIUM	Centralization Risks

Description:

In the TokimonsterRewarder.move module, the update_team_reward function grants the administrator unrestricted control to modify the team_reward parameter at any time.

For example, a user may participate in the system when team_reward is set to 10. However, during their interaction, the administrator could arbitrarily increase the value to 90. Since users are not notified or given time to react, this behavior may lead to financial losses and creates an opaque, centralized risk.

This pattern of unchecked administrative power undermines user trust and contradicts the decentralized ethos of blockchain systems.

```
public entry fun update_team_reward(signer: &signer, team_reward: u64) acquires RewarderConfig {
        let signer_addr = signer::address_of(signer);
        assert!(signer_addr == @Tokimonster, ENOT_TOKIMONSTER);
        let obj_address = get_obj_address();
        let rewarder = borrow_global_mut<RewarderConfig>(obj_address);
        rewarder.team_reward = team_reward;

        let event = UpdateTeamRewardEvent {
            operator: signer_addr,
            store_address: obj_address,
            team_reward,
        };
        emit(event);
    }
```

Recommendations:

Implement a Timelock mechanism to delay the effect of administrative updates. This allows users to exit or respond to changes before they take effect. Projects like Compound have successfully

adopted such mechanisms to safeguard governance functions. You can enforce a delay between scheduling and executing team_reward updates to mitigate centralization risk.

Result	FixResult
Confirmed	Acknowledge

3.2 Bypassing team_reward Limitation

Location	Severity	Category
TokimonsterRewarder.move	LOW	Business Logic Flaw

Description:

In the initialize function of TokimonsterRewarder, there is a constraint ensuring team_reward does not exceed 100. However, the set_override_team_rewards_for_token and add_user_reward_recipient functions lack similar limitations. If team_reward is set above 100, extract operations will fail due to insufficient funds, creating a potential vulnerability.

```
1
2 public entry fun initialize(signer: &signer, team_recipient: address, team_reward: u64) {
3     let signer_addr = signer::address_of(signer);
4     assert!(signer_addr == @Tokimonster, ENOT_TOKIMONSTER);
5     assert!(team_recipient != @0x0, ENOT_OWNER);
6     assert!(team_reward <= 100, EINVALID_AMOUNT);
```

Recommendations:

Add the same constraint (assert!(team_reward <= 100, ...)) in all functions that allow setting or overriding team_reward. This ensures uniform validation across the contract and prevents logic divergence.

Result	FixResult
Confirmed	Fixed

3.3 Invalid Address Replacement in replace_user_reward_recipient Function

Location	Severity	Category
TokimonsterRewarder.move	LOW	Business Logic Flaw

Description:

The replace_user_reward_recipient function in TokimonsterRewarder fails to verify whether old_recipient and recipient are the same. This allows for address - replacement operations that are, in reality, non - operational. Such actions result in unnecessary gas consumption and the generation of invalid event logs.

```
public entry fun replace_user_reward_recipient(signer: &signer, position: address, recipient: address) acquires RewarderStorage {
        let signer_addr = signer::address_of(signer);
        let obj_address = get_obj_address();
        let rewarder = borrow_global_mut<RewarderStorage>(obj_address);

        let old_recipient = if (rewarder.user_reward_recipient_for_token.contains(position)) {
            rewarder.user_reward_recipient_for_token.remove(position)
        } else {
            @0x0
        };
```

Recommendations:

Add a precondition check to short-circuit execution if old_recipient == recipient. This avoids no-op state changes and improves both gas efficiency and event clarity.

Result	FixResult
Confirmed	Fixed

3.4 Spelling Errors and Naming Conventions

Location	Severity	Category
	INFO	Code Quality

Description:

In the TokimonsterToken contract, the structure name ExtralMetadata contains a spelling error and should be corrected to ExternalMetadata. This inconsistency may compromise code readability and maintainability.

In the TokimonsterRewarder contract:

- The function name deposite_fa_to_user_and_team is misspelled and should be deposit_fa_to_user_and_team.

- The view functions get_postions_for_user and get_recipient_for_postion have spelling mistakes and should be renamed to get_positions_for_user and get_recipient_for_position respectively.

Recommendations:

Rename the three misnamed functions for clarity.

Result	FixResult
Confirmed	Fixed

4. CONCLUSION

In this audit, we thoroughly analyzed **tokimonsterAI** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

Contact

Website

www.exvul.com

Email

contact@exvul.com

Twitter

@EXVULSEC

Github

github.com/EXVUL-Sec

ExVul