Token & Vesting

Smart Contract Audit Report
Prepared for Moo Monster

Date Issued:	Dec 2, 2021
Project ID:	AUDIT2021049
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021049
Version	v1.0
Client	Moo Monster
Project	Token & Vesting
Auditor(s)	Weerawat Pawanawiwat
Author	Weerawat Pawanawiwat
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 2, 2021	Full report	Weerawat Pawanawiwat

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 Moo Monster, Inspex team conducted an audit to verify the security posture of the Token & Vesting smart contracts on Nov 29, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Token & Vesting 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 2 low, 1 very low, and 1 info-severity issues. With the project team's prompt response, 1 low, 1 very low, and 1 info-severity issues were resolved in the reassessment, while 1 low-severity issue was acknowledged by the team. Therefore, Inspex trusts that Token & Vesting smart contracts have sufficient protections to be safe for public use. However, 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

MooMonster is a Free-to-Play and Play-to-Earn NFT Game. The users can go on an adventure with the Moo monster in the "Mooniverse" world.

Token & Vesting smart contracts are responsible for the creation of $MOO, and the distribution of the token to the users in different categories according to the tokenomics. The token will be gradually released in steps, and the users can claim their token in each category through the MooVesting smart contract.

Scope Information:

Project Name	Token & Vesting
Website	https://moo-monster.com/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Nov 29, 2021
Reassessment Date	Dec 1, 2021

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: b41b575431843055677bfc2213ac42a4468760dc)

Contract	Location (URL)
MooMonsterToken	https://github.com/Moo-Monster/MooMonster-Contract/blob/b41b575431/contracts/tokens/MooMonsterToken.sol
MooVesting	https://github.com/Moo-Monster/MooMonster-Contract/blob/b41b575431/contracts/vesting/MooVesting.sol

Reassessment: (Commit: 2d07e927e5e8f66afea5cbdfb89b32ef74cfe385)

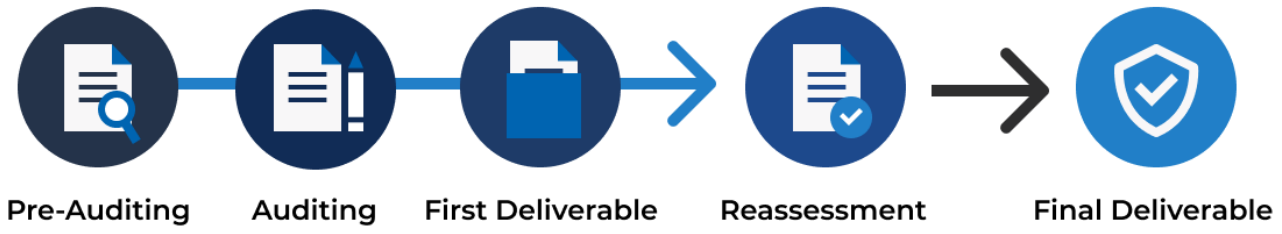
Contract	Location (URL)
MooMonsterToken	https://github.com/Moo-Monster/MooMonster-Contract/blob/2d07e927e5/contracts/tokens/MooMonsterToken.sol
MooVesting	https://github.com/Moo-Monster/MooMonster-Contract/blob/2d07e927e5/contracts/vesting/MooVesting.sol

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 following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] 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

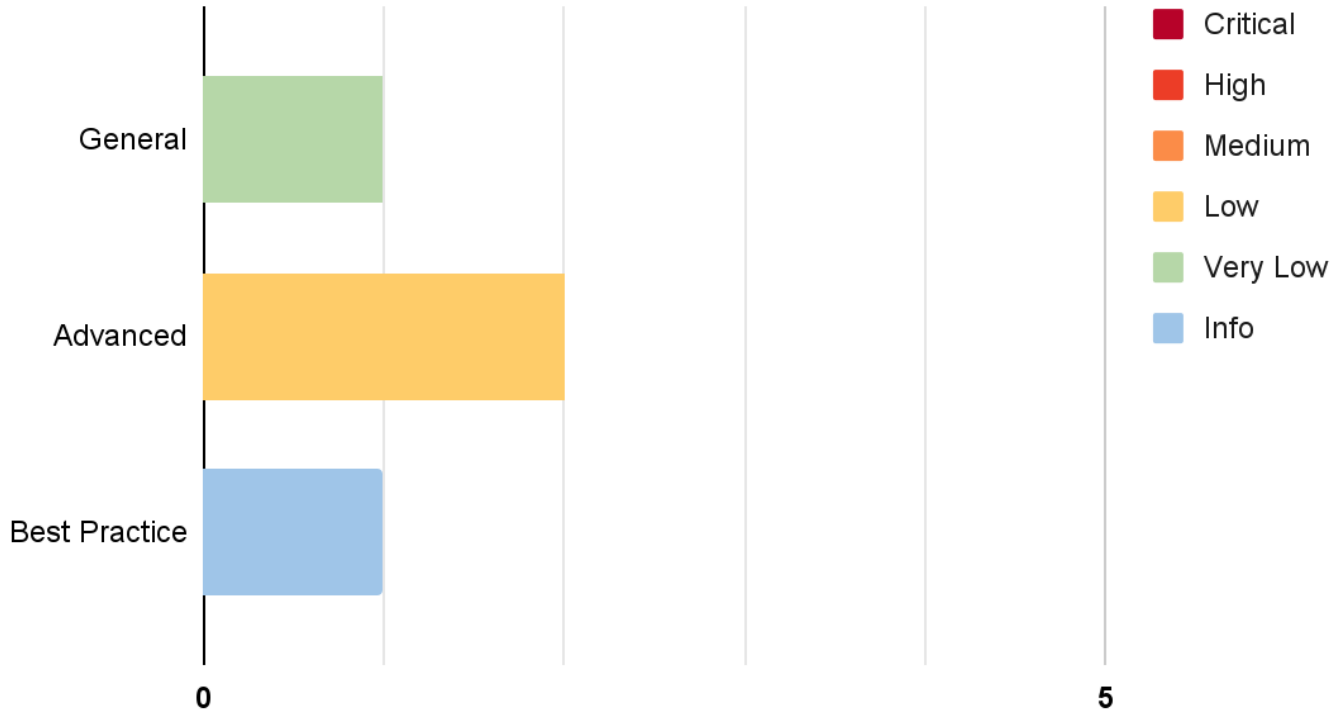
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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

From the assessments, Inspex has found 4 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



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	Improper Reward Calculation for Event Emission	Advanced	**Low**	Resolved
IDX-002	Token Withdrawal by Contract Owner	Advanced	**Low**	Acknowledged
IDX-003	Outdated Solidity Compiler Version	General	**Very Low**	Resolved
IDX-004	Inexplicit Solidity Compiler Version	Best Practice	**Info**	Resolved

5. Detailed Findings Information

5.1. Improper Reward Calculation for Event Emission

ID	IDX-001
Target	MooVesting
Category	Advanced Smart Contract Vulnerability
CWE	CWE-682: Incorrect Calculation
Risk	**Severity:** Low **Impact:** Low The amount of reward claimed in the event emitted will be incorrect. This disrupts the tracking of token amounts from logs, and may result in loss of reputation for the platform. **Likelihood:** Medium The miscalculation will happen when the reward is not claimed to the latest step before the claiming of the last step reward.
Status	Resolved Moo Monster team has resolved this issue as suggested in commit `2d07e927e5e8f66afea5cbdfb89b32ef74cfe385` by modifying the calculation logic.

5.1.1. Description

The `MooVesting` contract manages the distribution of $MOO by allowing the designated users to receive the token in steps. The claimable amount for each step is calculated using the percentage defined in each category. For the release of the last step, all of the remaining reward will be distributed to the user as seen in line 232.

MooVesting.sol

```
216 uint256 secondRelease = tgeTimestamp.add(category.cliffAfterTGE);
217 uint256 rewarded = alreadyRewarded[targetHash];
218 // claim reward after TGE
219
220 for (
221     uint256 i = lastClaimedStep[targetHash] + 1;
222     i <= category.totalSteps;
223     i++
224 ) {
225     uint256 addedAmount = 0;
226
227     if (secondRelease.add(category.stepTime.mul(i)) <= block.timestamp) {
228         lastClaimedStep[targetHash] = i;
229
```

```
230         if (i == category.totalSteps) {
231             // last step release all
232             addedAmount = _amount.sub(rewarded);
233         } else {
234             addedAmount = _amount.mul(category.percentAfter).div(100_00);
235         }
236
237         reward = reward.add(addedAmount);
238
239         emit StepClaim(_target, _category, i, addedAmount, block.timestamp);
240     } else {
241         break;
242     }
243 }
```

However, if the `rewarded` variable is not updated to the latest step before the last, the value of `addedAmount` will exceed the intended amount. This causes the `StepClaim` event to have an improper value of `addedAmount` amount emitted, which can cause confusions for the people who are monitoring the logs.

Furthermore, the reward amount to be claimed in the `reward` variable will be overly inflated, but fortunately, the code at line 250-254 has capped the upper limit of the user's eligible reward, so that limit will not be exceeded.

MooVesting.sol

```
245 require(reward > 0, "MOOVesting: no tokens to claim");
246
247 uint256 resultReward = 0;
248
249 // if reward overlimit (security check)
250 if (rewarded.add(reward) > _amount) {
251     resultReward = _amount.sub(
252         rewarded,
253         "MOOVesting: no tokens to claim (security check)"
254     );
255 } else {
256     resultReward = reward;
257 }
```

5.1.2. Remediation

Inspex suggests modifying the calculation on line 232 to include the reward amount claimed in the current execution, for example:

MooVesting.sol

```
220 for (
221     uint256 i = lastClaimedStep[targetHash] + 1;
222     i <= category.totalSteps;
223     i++
224 ) {
225     uint256 addedAmount = 0;
226 
227     if (secondRelease.add(category.stepTime.mul(i)) <= block.timestamp) {
228         lastClaimedStep[targetHash] = i;
229 
230         if (i == category.totalSteps) {
231             // last step release all
232             addedAmount = _amount.sub(rewarded.add(reward));
233         } else {
234             addedAmount = _amount.mul(category.percentAfter).div(100_00);
235         }
236 
237         reward = reward.add(addedAmount);
238 
239         emit StepClaim(_target, _category, i, addedAmount, block.timestamp);
240     } else {
241         break;
242     }
243 }
```

5.2. Token Withdrawal by Contract Owner

ID	IDX-002
Target	MooVesting
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Medium The users will not be able to claim some parts of the token that they are eligible for, resulting in monetary loss for the users and reputation damage to the platform. **Likelihood:** Low The withdrawal can only be done 41 months after the start of the release period. So it is unlikely that the users will not be able to claim the token in time.
Status	**Acknowledged** Moo Monster team has acknowledged this issue. This functionality is required because some users have not verified their vesting addresses before the deployment of the `MooVesting` contract, so Moo Monster team needs this function to recover the unclaimed token after the vesting time, and 41 months gap is left for the users to have time to claim their tokens. Furthermore, 24 hours timelock is implemented in the contract to allow the users to monitor for the execution of this function. Therefore, the users will have sufficient time to act before the actual withdrawal of the token.

5.2.1. Description

In the `MooVesting` contract, the contract owner (`DEFAULT_ADMIN_ROLE`) can use the `queueEmergencyWithdraw()` and `emergencyWithdraw()` functions to withdraw the whole token balance from the contract.

The `queueEmergencyWithdraw()` function is allowed to be executed after 41 months from the start of the token distribution. It is used to set the `executeTime` state to be 24 hours after the execution of this function in order to queue for the execution of the `emergencyWithdraw()` function.

MooVesting.sol

```
312 function queueEmergencyWithdraw() external onlyRole(DEFAULT_ADMIN_ROLE) {
313     // Emergency
314     require(
315         block.timestamp >= tgeTimestamp + (41 * 30 days), // emergencyWithdraw
    can be also executed after 41 months
316         "MOOVesting: Emergency withdraw should perform 41 months after TGE"
317     );
```

```
318         executeTime = block.timestamp + 24 hours; // Timelock 24 hours
319         emit QueueEmergencyWithdraw(executeTime);
320     }
```

The `emergencyWithdraw()` function checks that the current time must be after the queued `executeTime` and transfers the whole balance of the token in the contract to the caller of the function.

MooVesting.sol

```
322 function emergencyWithdraw() external onlyRole(DEFAULT_ADMIN_ROLE) {
323     require(
324         executeTime != 0 && block.timestamp >= executeTime,
325         "MOOVesting: Invalid Timelock"
326     );
327 
328     uint256 totalAmount = IERC20(token).balanceOf(address(this));
329     IERC20(token).safeTransfer(_msgSender(), totalAmount);
330 
331     executeTime = 0;
332 
333     emit EmergencyWithdraw(_msgSender(), totalAmount);
334 }
```

After the withdrawal by the contract owner, the users will not be able to withdraw their token anymore as there will be insufficient token balance in the contract.

5.2.2. Remediation

Inspex suggests removing the contract owner's ability to withdraw the token after the start of the token distribution time.

5.3. Outdated Solidity Compiler Version

ID	IDX-003
Target	MooMonsterToken
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves. **Likelihood:** Low From the list of known Solidity bugs, it is very unlikely that those bugs would affect this smart contract.
Status	Resolved Moo Monster team has resolved this issue as suggested in commit `2d07e927e5e8f66afea5cbdfb89b32ef74cfe385` by setting the Solidity compiler version to 0.8.10.

5.3.1. Description

The Solidity compiler version specified in the smart contract was outdated. This version has publicly known inherent bugs that may potentially be used to cause damage to the smart contract or the users of the smart contract.

MooMonsterToken.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.9;
```

5.3.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version. During the audit activity, the latest stable version of Solidity compiler in major 0.8 is v0.8.10[2]. For example:

MooMonsterToken.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.10;
```

5.4. Inexplicit Solidity Compiler Version

ID	IDX-004
Target	MooVesting
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Moo Monster team has resolved this issue as suggested in commit `2d07e927e5e8f66afea5cbdfb89b32ef74cfe385` by setting the Solidity compiler version to 0.8.10.

5.4.1. Description

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

MooVesting.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.9;
```

5.4.2. 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 v0.8.10[2]. For example:

MooVesting.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.10;
```

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

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] “Release Version 0.8.10 · ethereum/solidity.” [Online]. Available: https://github.com/ethereum/solidity/releases/tag/v0.8.10. [Accessed: 01-Dec-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE