SALUS SECURITY

SEP 2025

CODE SECURITY ASSESSMENT

RICEAI

Overview

Project Summary

- Name: RiceAI Token
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/0xdeveloperdegen/contracts
- Address:
  - RiceAITokenV2: 0xb5761f36FdFE2892f1b54Bc8EE8baBb2a1b698D3
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	RiceAI Token
Version	v2
Type	Solidity
Dates	Sep 16 2025
Logs	Sep 15 2025; Sep 16 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	8
Total	12

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Mint will always revert by transferring erc721 to the zero address	High	Business Logic	Resolved
2	Missing State Change Validation	Low	Configuration	Resolved
3	Lack of testing	Low	Configuration	Resolved
4	Incomplete Docstrings	Informational	Code Quality	Resolved
5	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Resolved
6	Non-explicit Imports Reduce Code Readability	Informational	Code Quality	Resolved
7	Mapping Declaration Lacks Named Parameters for Improved Readability	Informational	Code Quality	Resolved
8	Duplicate Imports	Informational	Redundancy	Resolved
9	Missing two-step transfer ownership pattern	Informational	Business Logic	Resolved
10	Gas Optimization	Informational	Gas Optimization	Resolved
11	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Mint will always revert by transferring erc721 to the zero address

Severity: High	Category: Business Logic
Target: - contracts/RiceRoboticsNFTV2.sol	

Description

contracts/RiceRoboticsNFTV2.sol:L69-L83

```
function mint(uint256 _nftId) external payable {
    require(_nftId > 0, "Must mint at least one token");
    require(
        riceAINftV1.ownerOf(_nftId) == msg.sender,
        "Must own the Rice AI NFT"
    );

    riceAINftV1.transferFrom(msg.sender, address(0), _nftId);
    uint256 startTokenId = totalSupply() + 1;
    _mint(msg.sender, startTokenId);

    emit TokensMinted(msg.sender, 1, startTokenId);
}
```

The `mint()` function attempts to “burn” the V1 NFT by transferring it to `address(0)`, which violates the `ERC-721` standard and always reverts, blocking the entire V1→V2 migration flow.

openzeppelin-contracts/blob/v5.0.1/contracts/token/ERC721/ERC721.sol

```
function transferFrom(address from, address to, uint256 tokenId) public virtual {
    if (to == address(0)) {
        revert ERC721InvalidReceiver(address(0));
    }
    // Setting an "auth" arguments enables the `_isAuthorized` check which verifies that
the token exists
    // (from != 0). Therefore, it is not needed to verify that the return value is not 0
here.
    address previousOwner = _update(to, tokenId, _msgSender());
    if (previousOwner != from) {
        revert ERC721IncorrectOwner(from, tokenId, previousOwner);
    }
}
```

OpenZeppelin `ERC-721` (v5.x) reverts with zero-address receivers with `ERC721InvalidReceiver(address(0))`, therefore no V2 token can ever be minted and users cannot migrate.

Recommendation

It is recommended to change to a more reasonable method for destroying V1 tokens.

Status

The team has resolved this issue in commit 4e129d9.

2. Missing State Change Validation

Severity: Low	Category: Configuration
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

Throughout the codebase, multiple instances of functions that do not verify whether the new value actually differs from the existing one before updating were identified:

- The `setTransferAllowedTimestamp()` function in RiceAITokenV2.sol
- The `addToWhitelist()` function in RiceAITokenV2.sol
- The `removeFromWhitelist()` function in RiceAITokenV2.sol
- The `markBotDelivered()` function in RiceRoboticsNFTV2.sol

Recommendation

Consider adding validation checks that revert the transaction if the input value matches the existing value.

Status

The team has resolved this issue in commit 0838db3 and 0531368.

3. Lack of testing

Severity: Low	Category: Configuration
Target: - test/RiceNFT.test.ts	

Description

Currently, while a test file is present, it lacks any meaningful content, and the existing functionalities have not been validated against the expected behavior. This absence of proper testing increases the risk of undiscovered defects or regressions, which in turn may compromise the overall security, reliability, and maintainability of the contract. Comprehensive and well-structured test coverage is strongly recommended to ensure robust verification of critical functionalities.

Recommendation

It is recommended to increase the test coverage.

Status

The team has resolved this issue in commit 1772077.

2.3 Informational Findings

4. Incomplete Docstrings

Severity: Informational	Category: Code Quality
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

In the `RiceAI` and `RiceRoboticsNFTV2` contracts ,multiple instances of incomplete docstrings were identified across the codebase. Several functions/events do not document all of their parameters or return values. This reduces readability and maintainability of the code, and increases the effort required by developers, auditors, or integrators to fully understand the contract logic. When such functions are part of the contract's public API, incomplete documentation may further hinder external usage and integration.

Recommendation

It is recommended to provide complete documentation for all public functions and events, including details on their parameters and return values. Following the Ethereum Natural Specification Format (NatSpec) is highly encouraged to ensure consistency, readability, and usability. Comprehensive documentation improves long-term maintainability and facilitates both auditing and integration processes.

Status

The team has resolved this issue in commit 739e69b and 756dfe2.

5. Lack of Security Contact Information for Responsible Disclosure

Severity: Informational	Category: Configuration
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

These contracts do not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

The team has resolved this issue in commit 97dab40 and 1c2b41a.

6. Non-explicit Imports Reduce Code Readability

Severity: Informational	Category: Code Quality

Target:
- contracts/RiceAITokenV2.sol
- contracts/RiceRoboticsNFTV2.sol

Description

`RiceAITokenV2.sol` and `RiceRoboticsNFTV2.sol` use wildcard or global-style import statements such as import "<path>";, which introduce all symbols from the imported file into the current compilation unit. While functional, this approach can reduce code readability and make it unclear which specific contracts, interfaces, or types are actually being used in the file. Explicit import { A, B } from "<path>"; declarations are generally preferred, as they make dependencies explicit and reduce the potential for namespace conflicts or unintentional symbol usage.

Recommendation

Consider refactoring a complete import statement to use named import syntax.

Status

The team has resolved this issue in commit b10f32a and 72f002a.

7. Mapping Declaration Lacks Named Parameters for Improved Readability

Severity: Informational	Category: Code Quality
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In the `RiceAITokenV2.sol` and `RiceRoboticsNFTV2.sol`, the mappings `whitelist` and `botDelivered` are declared without named parameters, making the purpose of the key and value less obvious upon inspection.
contracts/RiceAITokenV2.sol:L11

```
mapping(address => bool) public whitelist;
```

contracts/RiceRoboticsNFTV2.sol:L19

```
mapping(uint256 => bool) public botDelivered;
```

Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

Status

The team has resolved this issue in commit 9e188ea and b04ed05.

8. Duplicate Imports

Severity: Informational	Category: Redundancy
Target: - contracts/RiceRoboticsNFTV2.sol	

Description

In the `RiceRoboticsNFTV2` contract, a duplicate import instance was found:

The `RiceRoboticsNFTV2.sol` file imports `ERC721.sol` which is already imported as a result of importing `ERC721Enumerable.sol`.

Recommendation

It is recommended to remove duplicate import statements and explicitly import `ERC721` while importing `ERC721Enumerable` to improve the overall clarity and readability of the code base.

Status

The team has resolved this issue in commit 3e0e430.

9. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

The `RiceAI` and `RiceRoboticsNFTV2` contracts inherit from the `Ownable` contract. These contracts do not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 3e0e430 and ada89a0.

10. Gas Optimization

Severity: Informational	Category: Gas Optimization
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

In `RiceAI` and `RiceRoboticsNFTV2`, the contracts use <require(condition, "error message") statements>. Since Solidity version 0.8.26, require statements support custom errors, which are more gas-efficient and improve code clarity. Initially, this feature was only available through the IR pipeline, but starting from Solidity 0.8.27, it is also supported in the legacy pipeline.

Recommendation

Consider replacing all require(condition, "error message") statements with require(condition, CustomError()) to improve readability and reduce gas consumption.

Status

The team has resolved this issue in commit 1488ca3 and 2186a46.

11. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - contracts/RiceAITokenV2.sol - contracts/RiceRoboticsNFTV2.sol	

Description

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.28;
```

The `RiceAI` uses a floating compiler version `^0.8.0` and `RiceRoboticsNFTV2` uses `^0.8.28`.

Using floating pragmas `^0.8.0` and `^0.8.28` statements is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 49508d3 and 27bc1b3.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 00ac3b5:

File	SHA-1 hash
contracts/RiceAITokenV2.sol	fabf3b345a167ab49113e3e0688e940886fa54fc
contracts/RiceRoboticsNFTV2.sol	62c160b157bb5720e795fae344032da662a10417