Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **RabbitSwap**

Version: 1.0 (Final report)
Date: 9th May 2025
Commit/Hash: 00cf31911dc0e407ae40efbca014ae5ed7fa1f75
Auditor: Anh Bui
Reviewer: Duc Cuong Nguyen

Contents

1 Introduction

1.1 Project Overview

`RabbitSwap` is a next-generation decentralized exchange (DEX), forked from Uniswap V3, designed to deliver enhanced efficiency and improve user experience. A distinct feature of `RabbitSwap` is its ZeroGas mechanic on the Viction network, enabling users to execute transactions without incurring gas fees under specific conditions. This innovation significantly reduces trading costs, making decentralized trading more accessible to a wider audience.

Building upon this innovation, `RabbitSwap` further integrates a Sponsored Farm Feature, where users can stake their NFT-based liquidity provider positions to earn rewards. By leveraging upgradeable contract patterns, the system remains flexible for future enhancements.

1.2 Objective

This security audit report was provided by `CYBRING` on 10th April 2025. This audit was conducted to assess the robustness, reliability, and security of the code base. The goal was to identify vulnerabilities, ensure compliance with best security practices, and provide mitigation measures.

After the initial audit report, a re-assessment was performed on 9th May 2025 to verify the status of the reported issues. While most issues were addressed, a few remain open for future release.

1.3 Disclaimer

This security audit is not produced to replace any other type of assessment and does not aim to guarantee the discovery of all security issues within the scope of the assessment. While this audit was carried out with good faith and technical proficiency, it's crucial to understand that no single audit can guarantee the absolute security of a smart contract. To minimize risks, `CYBRING` recommends a multi-faceted approach involving multiple independent evaluations. Please note that this report is not intended to provide financial advice.

2 Scope of Audit

This audit is a follow-up engagement focused on the new Sponsored Farm Feature introduced after RabbitSwap's core platform underwent a prior comprehensive audit. Specifically, this assessment covers the farming logic, including the yield-farming contracts and signature-verification reward mechanisms. All other RabbitSwap components remain outside the scope of this follow-up audit.

The audit was conducted on commit 00cf31911dc0e407ae40efbca014ae5ed7fa1f75 from git repository `https://github.com/RabbitDEX/rabbitswap-contracts` Further details regarding The Sponsored Farm Feature of `RabbitSwap` audit scope are provided below:

- **Smart contracts audited:** See Appendix.

- **Codebase details:**
  - Language: Solidity
  - Frameworks: Hardhat
  - Initial audit's commit hash: ecfd525307cf7495f751d78ade3d9eda64fc32bb
  - Reassessment's commit hash: 00cf31911dc0e407ae40efbca014ae5ed7fa1f75

3 Audit Summary

Severity	Count
Medium	2
Low	1
Informational	4

4 Methodology

`CYBRING` conducts the following procedures to enhance security level of our client's smart contracts:

- **Pre-auditing:** Understanding the business logic of the smart contracts, investigating the deployment states of samples, and preparing for the audit.
- **Auditing:** Examining the smart contract by evaluating on multiple perspectives:
  - **Manual code review:** `CYBRING` auditors evaluate the static code analysis report for finding false positive reports. Besides, inspect the smart contract logic, access controls, and data flows ensure the contract behaves as intended and is free from risky logic.
  - **Static code analysis:** By using advanced static analysis techniques combined with our customized detectors, we set out to identify potential vulnerabilities, optimize gas usage, and ensure adherence to best practices in smart contract development.
  - **Fuzz testing:** `CYBRING` leverages fuzzing tools to stress-test the contract, ensuring it performs securely under unpredictable conditions.
- **First deliverable and consulting:** Presenting an initial report on the findings with recommendations for remediation and offering consultation services.
- **Reassessment:** Verifying the status of the issues and identifying any additional complications in the implemented fixes.
- **Final deliverable:** Delivering a comprehensive report detailing the status of each issue.

4.1 Audit Categories

- **Common vulnerabilities:** Smart contracts are analyzed following OWASP smart contract top 10 and Smart Contract Weakness Classification (SWC).
- **Advanced vulnerabilities:** `CYBRING` simulates a certain types of attack scenarios to exploit the smart contracts. These scenarios were prioritized from high to low severity.
- **Security best practices:** The source code of the smart contract is analyzed from the development perspective, providing suggestions for improving the overall code quality.

4.2 Audit Items

Table 1 show the details of the issues our auditors will conduct the audit upon.

Category	**Item**
Reentrancy attack	Single function reentrancy
	Reentrancy via modifier
	Cross function reentrancy
	Cross contract reentrancy
Data Validation and Integrity	Integer overflow and underflow
	Missing zero address validation
	Builtin symbols, state variables, and function should not be shadowed
	Storage variables should be initialized at the time of declaration
Denial of service (DOS)	Denial of service with revert
	Denial of service with gas limit
	Denial of service with induction variable overflow
	Denial of service by exceeding the maximum call stack depth
Access control vulnerabilities	Unprotected access to call sensitive function (self-destruct/suicide)
	Dangerous usage of tx.origin
Private information and randomness	Store sensitive information on smart contracts
	Generate random numbers from insecure pseudorandom factors
Timestamp dependence	Timestamp dependence on critical function
Best practices	Missing event for changing critical access control
	Follow standard style guide (naming conversion, code layout, order of layout) from Solidity

Table 1: Details of examined items

4.3 Risk Rating

The OWASP Risk Rating Methodology was applied to assess the severity of each issue based on the following criteria, arranged from the perspective of smart contract security.

- **Likelihood:** a measure of how likely this vulnerability is to be discovered and exploited by an attacker.
- **Impact:** a measure of the potential consequences or severity of a vulnerability if it is exploited by an attacker, including the extent of damage, data loss, or disruption of operations.

Impact	*High*	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Informational	Low	Medium
		Low	*Medium*	*High*
		Likelihood		

Table 2: Overall Risk Severity

5 Detailed Findings

5.1 Scalability and Maintainability Risks

- **Description:** By declaring state variables as *public* with *override*, the compiler automatically creates getter functions that are part of the contract's interface. However, this prior implementation can lead to various upgradability issues when the storage layout is changed or backward compatibility needs to be maintained during upgrades.
- **Risk:** Medium
- **CWE:** CWE-710: Improper Adherence to Coding Standards
- **Affected code:**
  - *contracts/farm/RabbitSponsoredFarm.sol*
  - *contracts/farm/interfaces/IRabbitSponsoredFarm.sol*
- **Mitigation:** `CYBRING` recommends changing the visibility of state variables from *public* to *private* and updating the interface and getter functions accordingly.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.2 Missing Deactivate Farm Mechanism

- **Description:** The property *active* in *Farm* structure is used to check Farm's state in many functions e.g., *harvest()*, *depositReward()*, ... However, the active value is always true after the farm registration.
- **Risk:** Medium
- **CWE:** CWE-840: Business Logic Errors.
- **Affected code:** *contracts/farm/RabbitSponsoredFarm.sol*
- **Mitigation:** Implement the Deactivate Farm Mechanism to fulfill smart contract business logic.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.3 Outdated OpenZeppelin Library Version

- **Description:** The OpenZeppelin Library used in the Sponsored Farm Feature is *3.4.2-solc-0.7* which is outdated and has many security-related issues, e.g., CVE-2022-39384, CVE-2022-35915.
- **Risk:** Low
- **CWE:** CWE-937: Using Components with Known Vulnerabilities.
- **Affected code:**
  - *contracts/farm/RabbitSponsoredFarm.sol*
  - *contracts/farm/interfaces/IRabbitSponsoredFarm.sol*
- **Mitigation:** `CYBRING` recommends using the latest OpenZeppelin Contracts Library (5.3.0) with Solidity compatibility compiler version.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.4 Checks-Effects-Interactions Pattern Not Followed in Implementation

- **Description:** By not following checks-effects-interactions, the below functions are vulnerable to Re-entrancy attacks. However, implementing these functions with *onlyOwner* and *nonReentrant* modifiers mitigates various attack surfaces.
- **Risk:** Informational
- **CWE:**
  - CWE-398: Indicator of Poor Code Quality
  - CWE-710: Improper Adherence to Coding Standards

- **Affected code:**
  - *contracts/farm/RabbitSponsoredFarm.sol:harvest()*
  - *contracts/farm/RabbitSponsoredFarm.sol:depositReward()*
- **Mitigation:** `CYBRING` recommends that the above functions be implemented, ensuring adherence to the checks-effects-interactions pattern.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.5 Outdated Solidity Compiler Version

- **Description:** The used solidity pragma in Sponsored Farm Feature is $\geq$ *0.7.6*. The compiler 0.7.6 was recognized with some security-relevant bugs (e.g., SOL-2023-2, SOL-2023-1).
- **Risk:** Informational
- **CWE:** CWE-937: Using Components with Known Vulnerabilities.
- **Affected code:**
  - *contracts/farm/RabbitSponsoredFarm.sol*
  - *contracts/farm/interfaces/IRabbitSponsoredFarm.sol*
- **Mitigation:** `CYBRING` recommends using a single, strict pragma version on all smart contracts (e.g., pragma solidity 0.8.29). This approach ensures that all contracts are compiled using the same compiler version, eliminating inconsistencies and compatibility issues.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.6 Redundant Function Visibility

- **Description:** Functions that are never called internally should not have public visibility, as this increases the contract's attack surface unnecessarily.
- **Risk:** Informational
- **CWE:** CWE-710: Improper Adherence to Coding Standards
- **Affected code:**

```
    function initialize(
        address _nonfungiblePositionManager
    ) public initializer {
        require(
            _nonfungiblePositionManager != address(0),
            'Invalid NFT manager'
        );
        nonfungiblePositionManager = INonfungiblePositionManager(
            _nonfungiblePositionManager
        );
        __Context_init();
        __Ownable_init();
        __ReentrancyGuard_init();
        __EIP712_init('RabbitSponsoredFarm', '1');
    }
```

contracts/farm/RabbitSponsoredFarm.sol:41-55

- **Mitigation:** Change public functions (which are never called internally) to external if they are only accessed from outside, or private/internal if exclusively used within the contract or inherited classes
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

5.7 Gas Optimization

- **Description:** In *contracts/farm/RabbitSponsoredFarm.sol:harvest()*, local *farm* variable was declared with *memory* data location. However, when the contract needs to update *totalClaimed*, it has to use a state variable. Overall, the prior approach is neither gas optimizing nor clearly implemented.
- **Risk:** Informational
- **CWE:** CWE-398: Indicator of Poor Code Quality
- **Affected code:**

```
        Farm memory farm = _farms[params.farmId];
        require(farm.active, 'Farm not active');

        bytes32 structHash = keccak256(
            abi.encode(
                HARVEST_TYPEHASH,
                params.tokenId,
                params.farmId,
                params.totalClaimable,
                params.blockNumber
            )
        );

        bytes32 digest = _hashTypedDataV4(structHash);

        require(
            digest.recover(params.signature) == farm.signer,
            'Invalid signature'
        );

        uint256 harvestAmount = params.totalClaimable -
            positionTotalClaimed[params.tokenId][params.farmId];
        require(harvestAmount > 0, 'No rewards to harvest');
        require(
            harvestAmount <= farm.totalClaimable - farm.
                totalClaimed,
            'Insufficient farm rewards'
        );

        positionLastHarvestBlock[params.tokenId] = block.number;
        positionTotalClaimed[params.tokenId][params.farmId] =
            params
             .totalClaimable;
        _farms[params.farmId].totalClaimed += harvestAmount;
```

contracts/farm/RabbitSponsoredFarm.sol:137-168

- **Mitigation:** Changing *farm* data location from *memory* to *storage*. Then using it for updating *totalClaimed* value.
- **Status [09/05/2025]:** The `RabbitSwap` team has addressed this issue, following the suggestion of `CYBRING`.

Appendix

```
contracts/
└─ farm/
   ├─ interfaces/
   │  └─ IRabbitSponsoredFarm.sol
   └─ RabbitSponsoredFarm.sol
```