Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Rabbitswap**

Version: 1.0 (final report)
Date: 11th December 2024
Commit/Hash: 14fea8e1232b89b5abd53e4f85d395bfeec48635
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.

1.2 Objective

This security audit report was provided by `CYBRING` on 3rd December 2024. 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 reassessment was conducted on 11th December 2024 to verify the status of the reported issues. While some issues were addressed, others were acknowledged and will be prioritized for future improvement.

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 assessments. Please note that this report is not intended to provide financial advice.

2 Scope of Audit

The audit was conducted on commit 14fea8e1232b89b5abd53e4f85d395bfeec48635 from git repository `https://github.com/RabbitDEX/rabbitv3-contracts` Further details regarding The RabbitSwap V3 audit scope are provided below:

- **Smart contracts audited:** See Appendix.
- **Codebase details:**
  - Language: Solidity
  - Frameworks: Hardhat
  - Initial audit's commit hash: 14fea8e1232b89b5abd53e4f85d395bfeec48635
  - Reassessment's commit hash: 8f464a9eb7aea03ad83061da4dd06af69b4c542b
- **Deploying state:** Table 1 shows the current address of the deployed smart contract on VIC SCAN..

Deploying Smart Contract	Address
WETH	0xc054751bdbd24ae713ba3dc9bd9434abe2abc1ce
RabbitSwapV3Factory	0xe615c973e92bdcb584bf9085c4612e342164e12a
RabbitSwapInterfaceMulticall	0x1a809bba84747fedd25296ccce00cf93aac3bc59
TickLens	0xadbcf57f452e48ae84d49ecc04f75422460eec91
QuoterV2	0x824649a6c3307f151205c253e99eaf8824faba4c
ProxyAdmin	0x887384cdb4c77109d88afafc54f7fc41defdfd9b
NFTDescriptorLibraryV1	0xf928a6ddf2dd2dc649753ad049c1de226154dbec
NFTPositionDescriptorV1	0x2dab37cf83dd58ae57fefa911e55ab98ad838095
NFTPositionDescriptorProxy	0x562e91c18cb892ef732b0736caee63e4ebc924ba
NFTPositionManager	0x383aa1a61066630b50e723fa7a3d22795e9e3179
SwapRouter	0x0a78ac5d4f3fff5b1b6bf884b4c75060d1994ee8

Table 1: Deploying smart contract address

3 Audit Summary

Severity	Count
Low	2
Informational	3

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, investigate the deploying states of samples and preparing for the auditing.
- **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 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 OSWAP 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. There 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 2 show the details of the issues our auditors will conduct the audit upon.

Category	**Item**
Reentrancy attack	Reentrancy attack Reentrancy via modifier Cross function reentrancy Cross contract reentracy
Integer overflow and under flow	Integer overflow Integer underflow
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 2: Details of examined items

4.3 Risk rating

The OWASP Risk Rating Methodology is 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 3: Overall Risk Severity

5 Detailed Findings

5.1 Timestamp dependence

- **Description:** Timestamp Dependence is a vulnerability in smart contracts where the contract's logic or behavior depends on the blockchain's timestamp, typically *block.timestamp*. This dependency can introduce security risks, as the timestamp can be manipulated by miners within certain limits.
- **Risk:** Low
- **CWE:** CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- **Affected code:**

```
232     function permit(address spender, uint256 tokenId, uint256
            deadline, bytes memory signature) external override {
233         require(deadline >= block.timestamp, 'VRC725: Permit
                deadline expired');
234         bytes32 digest = _getPermitDigest(spender, tokenId,
                _getNonce(tokenId), deadline);
```

contracts/vrc725/VRC725.sol:233

```
252     function permitForAll(address owner_, address spender, uint256
            deadline, bytes memory signature) external override {
253         require(deadline >= block.timestamp, 'VRC725: Permit
                deadline expired');
254         bytes32 digest = _getPermitForAllDigest(spender,
                _noncesByAddress[owner_], deadline);
```

contracts/vrc725/VRC725.sol:253

```
83          if (!initialized) {
84              (observationTimestamp, , , ) = IRabbitSwapV3Pool(pool).
                    observations(0);
85          }
86
87          secondsAgo = uint32(block.timestamp) - observationTimestamp
                ;
88      }
```

contracts/periphery/libraries/OracleLibrary.sol:87

```
102         (uint32 observationTimestamp, int56 tickCumulative, uint160
                secondsPerLiquidityCumulativeX128, ) =
103             IRabbitSwapV3Pool(pool).observations(observationIndex);
104         if (observationTimestamp != uint32(block.timestamp)) {
105             return (tick, IRabbitSwapV3Pool(pool).liquidity());
106         }
```

contracts/periphery/libraries/OracleLibrary.sol:104

- **Mitigation:** Avoid using *block.timestamp* for critical logic, consider adding a safe time buffer, consider using external oracles or multiple time sources for time-sensitive operation.
- **Status [11/12/2024]:** RabbitSwap team has acknowledged this issue.

5.2 Redundant expression

- **Description:** The redundant expression issue in smart contracts refers to the inclusion of unnecessary or repetitive code (or logic) within the contract. This potentially leads to inefficiencies, higher gas costs, and increased attack surface. Redundant expressions might cause security vulnerabilities, and make the contract harder to audit. In VRC7235.sol at line 854, the expression "*this;*" does not have any purpose and is included probably to suppress a compiler warning.
- **Risk:** Low
- **CWE:** CWE-695: Use of Low-Level Functionality
- **Affected code:**

```
853     function _getChainId() private view returns (uint256 chainId) {
854         this; // silence state mutability warning without
                    generating bytecode - see https://github.com/ethereum/
                    solidity/issues/2691
855         // solhint-disable-next-line no-inline-assembly
856         assembly {
857             chainId := chainid()
858         }
859     }
```

contracts/vrc725/VRC725.sol:854

- **Mitigation:** We recommend upgrading the Solidity compiler to version 0.8.0, eliminating the assembly block, and instead using *block.chainid* to enhance functionality and security.

- **Status [11/12/2024]:** The RabbitSwap team has acknowledged this issue.

5.3 Unused functions

5.3.1 Unused nonces

- **Description:** The *VRC725.nonces()* function is external, allowing it to be called by other contracts and through transactions.
- **Risk:** Informational
- **CWE:** CWE-710: Improper Adherence to Coding Standards.
- **Affected code:**

```
726     function nonces(uint256 tokenId) external view override returns
            (uint256) {
727         require(_exists(tokenId), "VRC725: invalid token ID");
728         return _nonces[tokenId];
729     }
```

contracts/vrc725/VRC725.sol:726-729

- **Mitigation:** To avoid potential unintended problems, it is recommended to remove this function.
- **Status [11/12/2024]:** The RabbitSwap team has acknowledged this issue.

5.3.2 Unused getAndIncrementNonce

- **Description:** The function *NonfungiblePositionManager._getAndIncrementNonce()* is currently unused in the contract. It appears to have been implemented without reference or invoke in any part of the code. Keeping unused functions can increase the contract's size, gas costs, and complexity, potentially creating confusion or unintended vulnerabilities.
- **Risk:** Informational.
- **CWE:** CWE-710: Improper Adherence to Coding Standards.
- **Affected code:**

```
382     function _getAndIncrementNonce(uint256 tokenId) internal
            returns (uint256) {
383         return uint256(_positions[tokenId].nonce++);
384     }
```

contracts/periphery/NonfungiblePositionManager.sol:382-384

- **Mitigation:** It is recommended to remove this function if it is not required or integrate it properly if it serves a future purpose.
- **Status [11/12/2024]:** The RabbitSwap team has addressed this issue, following `CYBRING`'s suggestion.

5.4 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:**

```
63      function enableFeeAmount(uint24 fee, int24 tickSpacing) public
            override {
64          require(msg.sender == owner);
65          require(fee < 1000000);
66          // tick spacing is capped at 16384 to prevent the situation
                where tickSpacing is so large that
67          // TickBitmap#nextInitializedTickWithinOneWord overflows
                int24 container from a valid tick
68          // 16384 ticks represents a >5x price change with ticks of
                1 bips
69          require(tickSpacing > 0 && tickSpacing < 16384);
70          require(feeAmountTickSpacing[fee] == 0);
71
72          feeAmountTickSpacing[fee] = tickSpacing;
73          emit FeeAmountEnabled(fee, tickSpacing);
74      }
```

contracts/core/RabbitSwapV3Factory.sol:63-74

```
26      function withdraw(uint256 wad) public payable {
27          require(balanceOf[msg.sender] >= wad);
28          balanceOf[msg.sender] -= wad;
29          (bool sent, ) = msg.sender.call{value: wad}(new bytes(0));
30          require(sent, 'Sent failed');
31          emit Withdrawal(msg.sender, wad);
32      }
```

contracts/external/WETH.sol:26-32

```
34      function totalSupply() public view returns (uint256) {
35          return address(this).balance;
36      }
```

contracts/external/WETH.sol:34-36

```
44      function transfer(address dst, uint256 wad) public returns (
            bool) {
45          return transferFrom(msg.sender, dst, wad);
46      }
```

contracts/external/WETH.sol:44-46

- **Mitigation:** These functions should have their visibility changed to external, which is specifically designed for external interactions. This change not only enhances security by limiting internal access, but also optimizes gas usage, as external functions are more efficient when called from outside the contract.

- **Status: 11/12/2024** The RabbitSwap team has fixed this issue, following `CYBRING`'s suggested mitigation.

5.5 Inexplicit solidity compiler version

- **Description:** 13 version pragmas are used across the smart contracts. This variation allows contracts to be compiled with different compiler versions, which can lead to potential compatibility issues.

  Upon analyzing the deployed contracts on `VICSCAN`, we found that all of them were compiled with Solidity version 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-664: Improper Control of a Resource Through its Lifetime.
  - CWE-937: Using Components with Known Vulnerabilities.

- **Mitigation:** To mitigate the potential compatibility issues caused by multiple pragma versions, we recommend using a single, strict pragma version across 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 [11/12/2024]:** The RabbitSwap team has acknowledged this issue.

Appendix

```
contracts/
├── core/RabbitSwapV3Factory.sol
├── core/RabbitSwapV3PoolDeployer.sol
├── core/RabbitSwapV3Pool.sol
│   ├── core/interfaces/IERC20Minimal.sol
│   ├── core/interfaces/IRabbitSwapV3PoolDeployer.sol
│   ├── core/interfaces/IRabbitSwapV3Factory.sol
│   └── core/interfaces/IRabbitSwapV3Pool.sol
│       ├── core/interfaces/callback/IRabbitSwapV3FlashCallback.sol
│       ├── core/interfaces/callback/IRabbitSwapV3MintCallback.sol
│       ├── core/interfaces/callback/IRabbitSwapV3SwapCallback.sol
│       ├── core/interfaces/pool/IRabbitSwapV3PoolActions.sol
│       └── core/interfaces/pool/IRabbitSwapV3PoolDerivedState.sol
├── vrc725/VRC725.sol
│   ├── vrc725/interfaces/IERC4494.sol
│   ├── vrc725/interfaces/IVRC725.sol
│   ├── vrc725/libraries/ECDSA.sol
│   ├── vrc725/libraries/ERC165.sol
│   └── vrc725/libraries/SignatrureChecker.sol
├── periphery/NonfungiblePositionManager.sol
├── periphery/NonfungibleTokenPositionDescriptor.sol
├── periphery/SwapRouter.sol
│   ├── periphery/base/LiquidityManagement.sol
│   ├── periphery/base/PeripheryPaymentsWithFee.sol
│   ├── periphery/interfaces/INonfungiblePositionManager.sol
│   ├── periphery/interfaces/ISwapRouter.sol
│   ├── periphery/lens/Quoter.sol
│   ├── periphery/lens/QuoterV2.sol
│   ├── periphery/lens/RabbitSwapInterfaceMulticall.sol
│   ├── periphery/libraries/AddressStringUtil.sol
│   ├── periphery/libraries/CallbackValidation.sol
│   ├── periphery/libraries/LiquidityAmounts.sol
│   ├── periphery/libraries/NFTDescriptor.sol
│   ├── periphery/libraries/NFTSVG.sol
│   ├── periphery/libraries/PoolAddress.sol
│   ├── periphery/libraries/PoolTicksCounter.sol
│   ├── periphery/libraries/PositionValue.sol
│   ├── periphery/libraries/SafeERC20Namer.sol
│   └── periphery/libraries/SqrtPriceMathPartial.sol
└── external/WETH.sol
```