AMM

Smart Contract Audit Report Prepared for QuickSwap

Date Issued:	May 12, 2022
Project ID:	AUDIT2021017
Version:	v2.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021017
Version	v2.0
Client	QuickSwap
Project	AMM
Auditor(s)	Suvicha Buakhom Patipon Suwanbol
Author	Suvicha Buakhom
Reviewer	Pongsakorn Sommalai
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	May 12, 2022	Change IDX-002 issue status	Suvicha Buakhom
1.0	Sep 17, 2021	Full report	Suvicha Buakhom

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 QuickSwap, Inspex team conducted an audit to verify the security posture of QuickSwap's AMM smart contracts between Aug 31, 2021 and Sep 3, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of QuickSwap's AMM 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 very low, and 1 info-severity issues. On the reassessment, 1 very lo-severity issue was resolved, while 1 very low and 1 info-severity issues were acknowledged by the team. The remaining risks of the issues found are acceptable. Therefore, Inspex trusts that QuickSwap's AMM 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

QuickSwap is a permissionless decentralized exchange (DEX) based on Ethereum, powered by Polygon Network's Layer 2 scalability infrastructure.

QuickSwap's AMM is an Automated Market Maker (AMM) protocol that is forked from Uniswap V2. On QuickSwap's AMM, users can perform ERC20 token swapping easily with the liquidity pool of the platform. Users can also provide liquidity to the pools and gain a part of the swapping fee and the platform's reward tokens.

Scope Information:

Project Name	AMM
Website	https://quickswap.exchange/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Aug 31, 2021 - Sep 3, 2021
Reassessment Date	Sep 17, 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:

Contract	Repository	Location (URL)
UniswapV2Factory	quickswap-core	https://github.com/QuickSwap/quickswap-core/blob/3cf503ebba/contracts/UniswapV2Factory.sol
UniswapV2Pair	quickswap-core	https://github.com/QuickSwap/quickswap-core/blob/3cf503ebba/contracts/UniswapV2Pair.sol
UniswapV2Router02	QuickSwap-periphery	https://github.com/QuickSwap/QuickSwap-periphery/blob/d28bf3428b/contracts/UniswapV2Router02.sol

Reassessment:

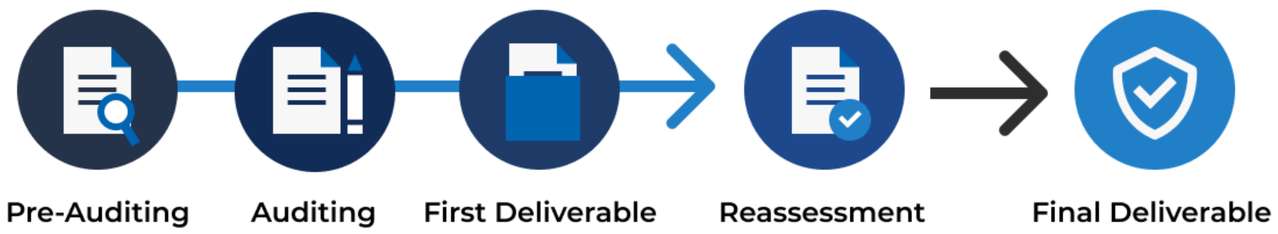
Contract	Repository	Location (URL)
UniswapV2Router02	QuickSwap-periphery	https://github.com/QuickSwap/QuickSwap-periphery/blob/522a94168b/contracts/UniswapV2Router02.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they are inherited 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
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Use of Upgradable Contract Design
Insufficient Logging for Privileged Functions
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

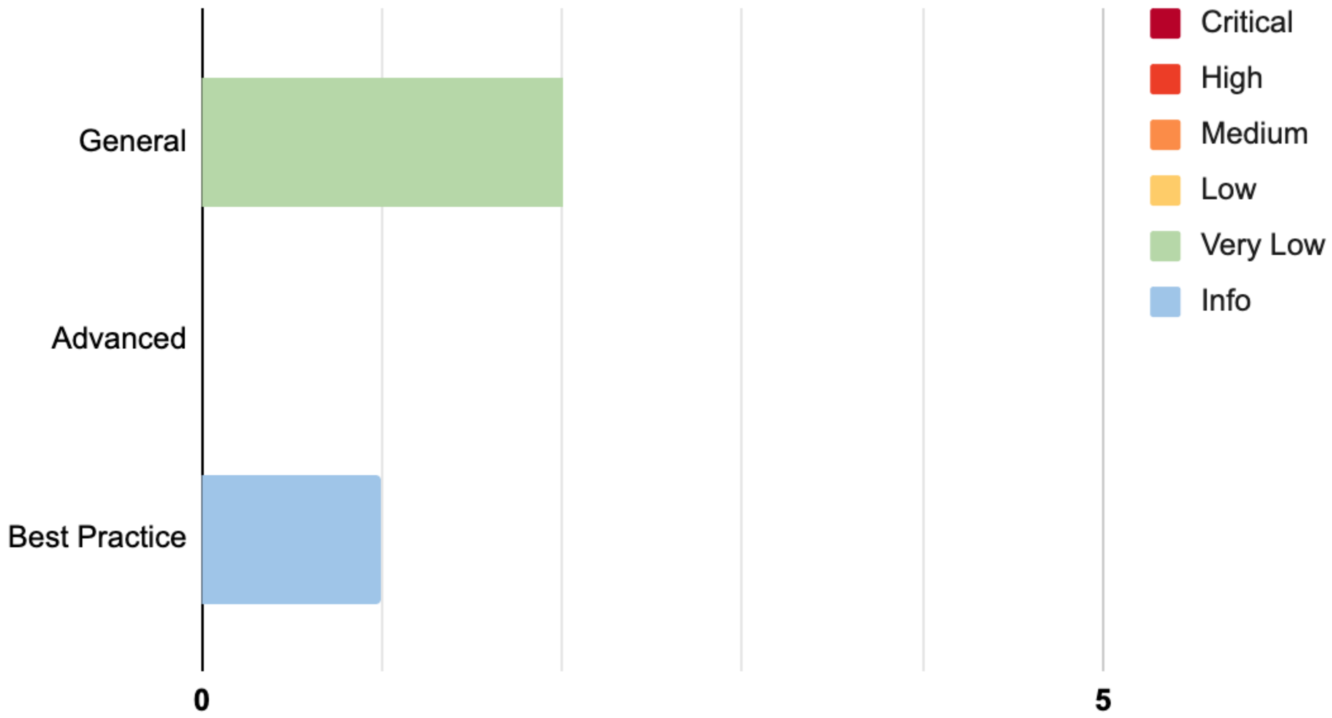
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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**.

Likelihood / Impact	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 3 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	Outdated Compiler Version	General	Very Low	Acknowledged
IDX-002	Invalid Address of Wrapped Native Token	General	Very Low	Resolved
IDX-003	Improper Function Visibility	Best Practice	Info	No Security Impact

* The mitigations or clarifications by QuickSwap can be found in Chapter 5.

5. Detailed Findings Information

5.1. Outdated Compiler Version

ID	IDX-001
Target	UniswapV2Factory UniswapV2Pair
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 these smart contracts.
Status	**Acknowledged** QuickSwap team has acknowledged this issue. As mentioned in the report, none of the bugs specified in the solidity version is going to impact the affected contracts.

5.1.1. Description

The Solidity compiler versions specified in the smart contracts were outdated. These versions have publicly known inherent bugs[2] that may potentially be used to cause damage to the smart contracts or the users of the smart contracts.

UniswapV2ERC20.sol

```
pragma solidity =0.5.16;
```

The following table represents contracts that use outdated versions of the compiler.

Contract	Version
UniswapV2Factory	0.5.16
UniswapV2Pair	0.5.16

5.1.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version[3].

During the audit activity, the latest stable versions of Solidity compiler in major 0.5 is v0.5.17

UniswapV2ERC20.sol

```
pragma solidity =0.5.17;
```

5.2. Invalid Address of Wrapped Native Token

ID	IDX-002
Target	UniswapV2Router02
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Very Low **Impact:** Low The transaction will be reverted and the user will lose the transaction fee. **Likelihood:** Low An error can occur only when the input token address is `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`, and it is very unlikely that this value will be used.
Status	Resolved QuickSwap team has applied the fix to commit `522a94168b0814d0776d834119df377f03898807`.

5.2.1. Description

The `checkAndConvertETHToWETH()` function in `UniswapV2Library` library, which is used in the `UniswapV2Router02` contract, converts the `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` address to the wrapped native token address ($WMATIC).

However, QuickSwap's contracts are designed to be used on the Polygon Network, and the `0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2` address is not the wrapped native token address ($WMATIC), but a wallet address instead.

UniswapV2Library.sol

```
function checkAndConvertETHToWETH(address token) internal pure returns(address)
{

    if(token == address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)) {
        return address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
    }
    return token;
}
```

The scenario below represents a situation when the `checkAndConvertETHToWETH()` function is executed.

When the router is called to swap tokens, it executes the `pairFor()` function to find the LP token address of tokens (`path[0]` and `path[1]`) in order to swap the tokens as shown in the `swapTokensForExactTokens()` function below.

UniswapV2Router02.sol

```
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint[] memory amounts) {
    amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
    require(amounts[0] <= amountInMax, 'UniswapV2Router: EXCESSIVE_INPUT_AMOUNT');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
    );
    _swap(amounts, path, to);
}
```

The `pairFor()` function calls the `sortTokens()` to find the correct order of tokens in order to get the correct hash value from `keccak256` hashing.

UniswapV2Library.sol

```
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address) {
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    return(address(uint(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1)),
hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f' // init code hash
        )))));
}
```

The `sortTokens()` function calls the `checkAndConvertETHToWETH()` function, passing the token address as a parameter to check and convert the predefined placeholder address for a native token to the wrapped native token address ($WMATIC).

UniswapV2Library.sol

```
function sortTokens(address tokenA, address tokenB) internal pure returns
```

```
(address, address) {
    tokenA = checkAndConvertETHToWETH(tokenA);
    tokenB = checkAndConvertETHToWETH(tokenB);
    return(tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA));
}
```

The `checkAndConvertETHToWETH()` function returns the incorrect wrapped native token address given `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` as a parameter. This could lead to an error when the swapping function, the `swapTokensForExactTokens()` function in this case, is executed.

5.2.2. Remediation

Inspex suggests using the correct address of `WMATIC` contract, in this case `0x0d500b1d8e8ef31e21c99d1db9a6444d3adf1270`, as the wrapped native token address on the Polygon Network.

UniswapV2Library.sol

```
function checkAndConvertETHToWETH(address token) internal pure returns(address)
{

    if(token == address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)) {
        return address(0x0d500b1d8e8ef31e21c99d1db9a6444d3adf1270);
    }
    return token;
}
```

5.3. Improper Function Visibility

ID	IDX-003
Target	UniswapV2Router02
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact QuickSwap team has acknowledged this issue.

5.3.1. Description

Functions with `public` visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `quote()` function of the `UniswapV2Router02` contract is set to `public` and it is never called from any internal function.

UniswapV2Router02.sol

```
403 function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual
    override returns (uint amountB) {
404     return UniswapV2Library.quote(amountA, reserveA, reserveB);
405 }
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

Contract	Function
UniswapV2Router02 (L:403)	quote()
UniswapV2Router02 (L:407)	getAmountOut()
UniswapV2Router02 (L:417)	getAmountIn()
UniswapV2Router02 (L:427)	getAmountsOut()
UniswapV2Router02 (L:437)	getAmountsIn()

5.3.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function as shown in the following example:

UniswapV2Router02.sol

```
function quote(uint amountA, uint reserveA, uint reserveB) external pure
virtual override returns (uint amountB) {
    return UniswapV2Library.quote(amountA, reserveA, reserveB);
}
```

6. Appendix

6.1. About Inspex

CYBERSECURITY
PROFESSIONAL
SERVICE

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] "List of Known Bugs — Solidity 0.5.16 documentation." [Online]. Available: https://docs.soliditylang.org/en/v0.5.16/bugs.html. [Accessed: 07-Sep-2021]

[3] ethereum, "Releases · ethereum/solidity." [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 07-Sep-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE