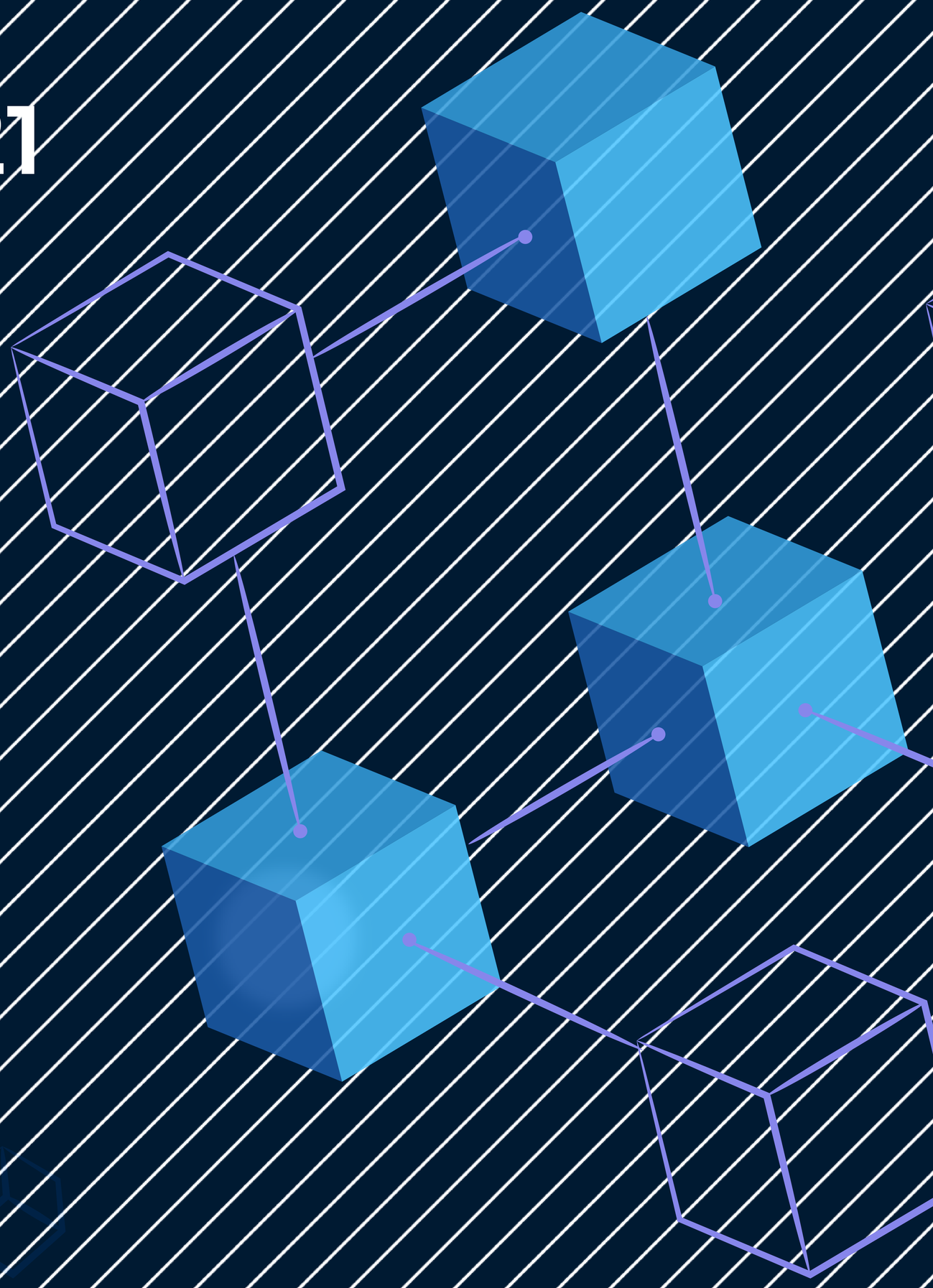
QuillAudits

Audit Report
December, 2021



For

FalconSwap

Contents

Scope of the Audit

The scope of this audit was to analyze and document the FalconSwap Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	1	0	0	0

Introduction

During the period of **November 05, 2021 to November 06, 2021** - QuillAudits Team performed a security audit for **FalconSwap** smart contracts.

The code for the audit was taken from following the official links:

Version 1 : https://bscscan.com/address/0x64e66afc3eac94be6547f7a21bee38567695cc43#code

Version 2: https://github.com/Falconswaps/Audit_Reports/blob/main/falcons.sol%205

Ver No	Date	Commit ID	Files
1	9 November	05d9b9e23181f21e8d9f47 170a4c552b001778b3	1. FalconSwap.sol

Issues Found

A. Contract – FalconSwap

High severity issues

1. Approve Race

Line	Code

755-766	function _transfer(
	address sender,
	address recipient,
	uint256 amount
	) internal {
	require(sender != address(0), 'BEP20: transfer from the zero address');
	require(recipient != address(0), 'BEP20: transfer to the zero address');
	
	_balances[sender] = _balances[sender].sub(amount, 'BEP20: transfer amount exceeds balance');
	_balances[recipient] = _balances[recipient].add(amount);
	emit Transfer(sender, recipient, amount);
	}

Description

On a transfer from user A to user B, the delegated votes should be updated according to the new balance of the users. But on transfer or transferFrom, no such modification in the _delegates mapping happens. If these delegated votes are not updated then the current vote balance of the accounts will not be correct.

Remediation

Override the _transfer function and call _moveDelegates in it.
Mint and burn function of the BEP20 contract will also need to be updated.

Status: Fixed

In version 2, the team fixed the code with the remediation provided above.

Medium Severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

No issues were found.

Functional Test Cases

- Should be able to transfer/transferFrom **PASS**
- Burn and burnFrom should decrease totalSupply **PASS**
- burnFrom and transferFrom should take allowance into consideration **PASS**
- Current delegate votes should be according to the current balance **PASS**
- Increase and decrease allowance should update allowance **PASS**
- Can delegate votes according to other account **PASS**

Automated Tests

Slither

```
Falcons._writeCheckpoint(address,uint32,uint256,uint256) (Falcons.sol#1052-1070) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (Falcons.sol#1062)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

BEP20.constructor(string,string).name (Falcons.sol#595) shadows:
        - BEP20.name() (Falcons.sol#611-613) (function)
        - IBEP20.name() (Falcons.sol#125) (function)
BEP20.constructor(string,string).symbol (Falcons.sol#595) shadows:
        - BEP20.symbol() (Falcons.sol#625-627) (function)
        - IBEP20.symbol() (Falcons.sol#120) (function)
BEP20.allowance(address,address).owner (Falcons.sol#659) shadows:
        - Ownable.owner() (Falcons.sol#63-65) (function)
BEP20._approve(address,address,uint256).owner (Falcons.sol#818) shadows:
        - Ownable.owner() (Falcons.sol#63-65) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Falcons.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (Falcons.sol#918-959) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,FLNS::delegateBySig: signature expired) (Falcons.sol#957)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (Falcons.sol#410-421) uses assembly
        - INLINE ASM (Falcons.sol#417-419)
Address._functionCallWithValue(address,bytes,uint256,string) (Falcons.sol#518-544) uses assembly
        - INLINE ASM (Falcons.sol#536-539)
Falcons.getChainId() (Falcons.sol#1077-1081) uses assembly
        - INLINE ASM (Falcons.sol#1079)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.4.22<0.9.0']
        - 0.6.12 (Falcons.sol#5)
        - >=0.4.22<0.9.0 (Migrations.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address._functionCallWithValue(address,bytes,uint256,string) (Falcons.sol#518-544) is never used and should be removed
Address.functionCall(address,bytes) (Falcons.sol#465-467) is never used and should be removed
Address.functionCall(address,bytes,string) (Falcons.sol#475-481) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Falcons.sol#494-500) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Falcons.sol#508-516) is never used and should be removed
Address.isContract(address) (Falcons.sol#410-421) is never used and should be removed
Address.sendValue(address,uint256) (Falcons.sol#439-445) is never used and should be removed
BEP20._burn(address,uint256) (Falcons.sol#796-802) is never used and should be removed
BEP20._burnFrom(address,uint256) (Falcons.sol#835-842) is never used and should be removed
Context._msgData() (Falcons.sol#27-30) is never used and should be removed
SafeMath.div(uint256,uint256) (Falcons.sol#304-306) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Falcons.sol#320-330) is never used and should be removed
SafeMath.min(uint256,uint256) (Falcons.sol#369-371) is never used and should be removed
SafeMath.mod(uint256,uint256) (Falcons.sol#344-346) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Falcons.sol#360-367) is never used and should be removed
SafeMath.mul(uint256,uint256) (Falcons.sol#278-290) is never used and should be removed
SafeMath.sqrt(uint256) (Falcons.sol#374-385) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Falcons.sol#439-445):
        - (success) = recipient.call{value: amount}() (Falcons.sol#443)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Falcons.sol#518-544):
        - (success,returndata) = target.call{value: weiValue}(data) (Falcons.sol#527)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable Falcons._delegates (Falcons.sol#860) is not in mixedCase
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Falcons.sol#28)" inContext (Falcons.sol#18-31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Falcons.sol#82-85)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Falcons.sol#91-93)
decimals() should be declared external:
        - BEP20.decimals() (Falcons.sol#618-620)
symbol() should be declared external:
        - BEP20.symbol() (Falcons.sol#625-627)
totalSupply() should be declared external:
        - BEP20.totalSupply() (Falcons.sol#632-634)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (Falcons.sol#651-654)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (Falcons.sol#659-661)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (Falcons.sol#670-673)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (Falcons.sol#687-699)
increaseAllowance(address,uint256) should be declared external:
        - BEP20.increaseAllowance(address,uint256) (Falcons.sol#713-716)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20.decreaseAllowance(address,uint256) (Falcons.sol#732-739)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (8 contracts with 75 detectors), 45 result(s) found
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **FalconSwap** platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **FalconSwap** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
December, 2021

For

FalconSwap

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com