

Audit Report September, 2021

For



Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity	03
Introduction	04
Issues Found – Code Review / Manual Testing	05
A. Contract - Falcon	05
High Severity Issues	05
Medium Severity Issues	05
Low Severity Issues	05
Informational Issues	05
Functional Tests	06
Automated Tests	07
Closing Summary	14

Scope of the Audit

The scope of this audit was to analyze and document the Falcon 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

Mythril, Slither, SmartCheck, Surya, Solhint.

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 four 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	0	0	0	0

Introduction

From **September 16, 2021, to September 17, 2021**, the QuillAudits Team performed a security audit for the Falcon smart contract.

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

[https://testnet.bscscan.com/
address/0xeCCE9653E17B3308667C5c8987CB57d776bB5788#code](https://testnet.bscscan.com/address/0xeCCE9653E17B3308667C5c8987CB57d776bB5788#code)



Issues Found

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

No issues were found.

Informational

No issues were found.



Functional Tests

Function Names	Testing results
constructor	PASS
symbol	PASS
name	PASS
totalSupply	PASS
balanceOf	PASS
Transfer	PASS
allowance	PASS
approve	PASS
transferFrom	PASS
increaseAllowance	PASS
decreaseAllowance	PASS

Automated Testing

SOLIDITY STATIC ANALYSIS

Gas & Economy

Gas costs:Gas requirement of function ERC20.name is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 61:4:

Gas costs:Gas requirement of function Falcons.name is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 61:4:

Gas costs:Gas requirement of function ERC20.symbol is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 69:4:

Gas costs:Gas requirement of function Falcons.symbol is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 69:4:

Gas costs:Gas requirement of function ERC20.transfer is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 112:4:

Gas costs:Gas requirement of function Falcons.transfer is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 112:4:

Gas costs:Gas requirement of function ERC20.allowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 120:4:

Gas costs:Gas requirement of function Falcons.allowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 120:4:

Gas costs:Gas requirement of function ERC20.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 131:4:

Gas costs:Gas requirement of function Falcons.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 131:4:

Gas costs:Gas requirement of function ERC20.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 149:4:

Gas costs:Gas requirement of function Falcons.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 149:4:

Gas costs:Gas requirement of function ERC20.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 177:4:

Gas costs:Gas requirement of function Falcons.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 177:4:

Gas costs:Gas requirement of function ERC20.decreaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 196:4:

Gas costs:Gas requirement of function Falcons.decreaseAllowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 196:4:

Miscellaneous

Constant/View/Pure functions:ERC20._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not.[more](#)Pos: 330:4:

Constant/View/Pure functions:ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not.[more](#)Pos: 350:4:

Constant/View/Pure functions:IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not.[more](#)Pos: 26:4:

Constant/View/Pure functions:IERC20.approve(address,uint256) : Potentially should be constant/view/pure but is not.[more](#)Pos: 51:4:

Constant/View/Pure functions:IERC20.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not.[more](#)Pos: 62:4:

Similar variable names:ERC20.(string,string) : Variables have very similar names "_name" and "name_".Pos: 54:8:

Similar variable names:ERC20.(string,string) : Variables have very similar names "_name" and "name_".Pos: 54:16:

Similar variable names:ERC20.(string,string) : Variables have very similar names "_symbol" and "symbol_".Pos: 55:8:

Similar variable names:ERC20.(string,string) : Variables have very similar names "_symbol" and "symbol_".Pos: 55:18:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 252:16:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 254:41:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 254:50:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 256:24:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 257:18:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 257:30:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 258:34:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 258:43:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 260:40:

Similar variable names:ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount".Pos: 260:49:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 275:16:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 277:29:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 277:50:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 279:43:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 280:34:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 282:22:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 282:50:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 284:24:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 286:22:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 286:43:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 288:28:

Similar variable names:ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount".Pos: 288:49:

No return:IERC20.totalSupply(): Defines a return type but never explicitly returns a value.Pos: 12:4:

No return:IERC20.balanceOf(address): Defines a return type but never explicitly returns a value.Pos: 17:4:

No return:IERC20.transfer(address,uint256): Defines a return type but never explicitly returns a value.Pos: 26:4:

No return:IERC20.allowance(address,address): Defines a return type but never explicitly returns a value.Pos: 35:4:

No return:IERC20.approve(address,uint256): Defines a return type but never explicitly returns a value.Pos: 51:4:

No return:IERC20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.Pos: 62:4:

No return:IERC20Metadata.name(): Defines a return type but never explicitly returns a value.Pos: 16:4:

No return:IERC20Metadata.symbol(): Defines a return type but never explicitly returns a value.Pos: 21:4:

No return:IERC20Metadata.decimals(): Defines a return type but never explicitly returns a value.Pos: 26:4:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 157:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 198:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 225:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 226:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 231:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 252:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 275:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 280:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 309:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 310:8:

Results

No major issue was 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. No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.



Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Falcon 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 Falcon Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.



Audit Report September, 2021

For



QuillAudits

📍 Canada, India, Singapore, United Kingdom

🌐 audits.quillhash.com

✉️ audits@quillhash.com