QuillAudits

Audit Report
November, 2023

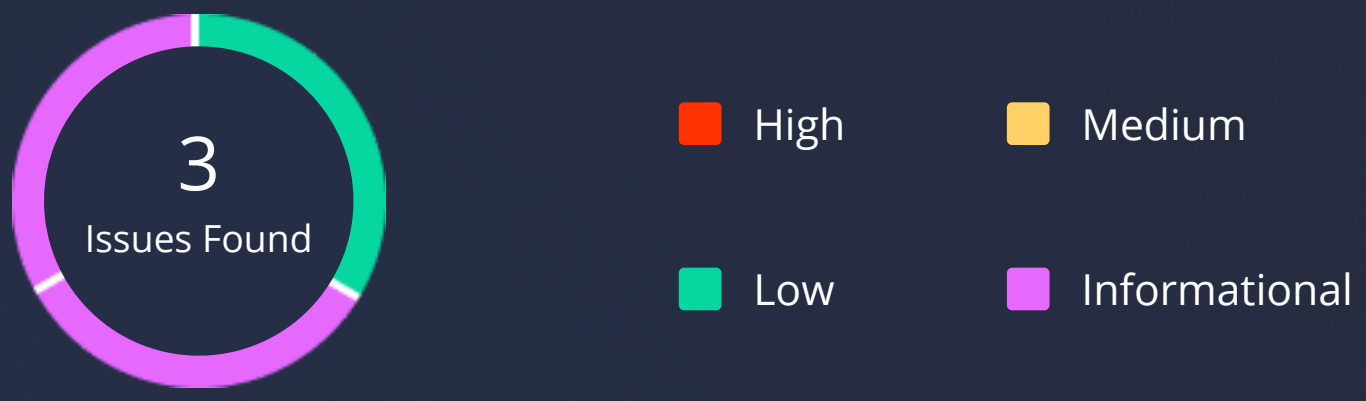
For

Table of Content

Executive Summary

Project Name	Falcons Inu
Overview	The FalconsInu token is a basic ERC20 token that adopts a mintingAllowance which allows the token owner to assign allowances to users to mint tokens before the supply cap is reached.
Timeline	28th November 2023 - 30th November 2023
Updated Code Received	1st December 2023
Second Review	1st December 2023
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the Falcons Inu codebase for quality, security, and correctness.
Source Code	*https://testnet.bscscan.com/token/0x817f69A833ad1330C082c9793d021075a734F913#code*
Fixed In	*https://bscscan.com/token/0xE038e4c8CEE0d2e63b0f077eC94a24C2B00Ac8d4#code*
Mainnet Address	*https://bscscan.com/address/0x87e2414093632a3b9a1afea7083e5dab59b5dc4f#code*

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	1	2

Checked Vulnerabilities

- 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 contracts, 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 analyzed 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

A 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 analyzed, 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 behavior 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

Hardhat, Foundry.

Types of Severity

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.

High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Low Severity Issues

1. Use Ownable2Step instead of Ownable

Path

https://testnet.bscscan.com/token/0x817f69A833ad1330C082c9793d021075a734F913#code

Function

renounceOwnership( ), transferOwnership( ), increaseMintingAllowance( ), decreaseMintingAllowance( ), and mint( )

Description

If ownership transfer is not properly done, the current contract owner can renounce/transfer ownership, lose owner privileges, and lose the ability to burn tokens. When ownership is renounced, all of the contract's methods listed above will be rendered unusable.

An edge cases could be:
- maxSupply is never reached (if ownership is renounced before minting allowances are granted or tokens get minted)

Recommendation

Functions such as renounceOwnership and transferOwnership can be overridden or set up for 2-step verification to prevent mistaken privilege transfer or renouncing.

References

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

and

https://github.com/razzorsec/RazzorSec-Contracts/blob/main/AccessControl/SafeOwn.sol

Status

Resolved

Informational Issues

2. Avoid function overloading via methods

Path

https://testnet.bscscan.com/token/0x817f69A833ad1330C082c9793d021075a734F913#code

Function

mint( )

Description

Although function overloading is supported in Solidity, it can be avoided in this contract as both mint functions are called by users of varying privilege. If a non-user calls mint( ) protected by onlyOwner, it could revert and fail thereby wasting gas.

Recommendation

The mint function that checks minting allowances can be renamed mintFromAllowance( ).

Status

Resolved

Falcon Inu Team's Comment

If we leave it as a mint(address to, uint256 amount), we will not make any issues in the future. Also, I believe it is not critical for the community or the exchanges.

3. Missing check for token minting/approval cap

Path

https://testnet.bscscan.com/token/0x817f69A833ad1330C082c9793d021075a734F913#code

Function

_approveMinters(...)

Description

There is no check that the maxSupply = totalSupply before granting subsequent users new minting allowances. Although upon token burns, the totalSupply gets reduced there is no guarantee that tokens will be burnt meaning that there would be no guarantee for users to mint new tokens - thus they would not need to have mintingAllowances set.

```
// approve the minting by the owner
function _approveMinters(address minter, uint256 amount) internal {    infinite gas
    require(minter != address(0), "BEP20: approve to the zero address");

    _mintingAllowances[minter] = amount;
    emit MintingApproval(_msgSender(), minter, amount);
}
```

Recommendation

Before users are granted approval for token minting, the contract can check if supply cap has been reached.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should deploy and mint 40% of tokens to deployer (owner)
- ✓ Should set mintingAllowance for new users
- ✓ Should use mintingAllowance to mint new tokens
- ✓ Should not allow new tokens be minted after supply cap is reached
- ✓ Should renounce and transfer ownership

Automated Tests

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

Closing Summary

In this report, we have considered the security of the Falcons Inu codebase. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Falcons Inu smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Falcons Inu smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Falcons Inu to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com