QuillAudits

AUDIT REPORT

March, 2025

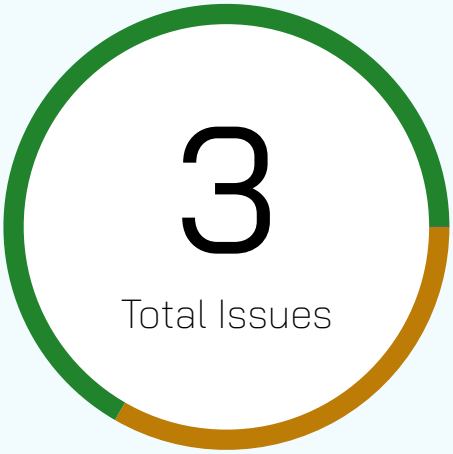
For

Table of Content

Executive Summary

Project name	EmberGuard
Project URL	https://www.snuffedwildfire.com/
Overview	EmberGuard (ticker: FIRE) is an ERC20 token with a fixed total supply of 10,000,000 tokens. It is built to drive social impact by funding wildfire mitigation and community initiatives.
Audit Scope	https://github.com/marcbcooper10/FireFund-Project-EmberGuard/tree/main/contracts
Contracts in Scope	BatchTransfer.sol, FireFund.sol, Migrations.sol
Commit Hash	5caaeba851e2743a9e0c5cf40f652917f9b3748a
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	17th March 2025 - 20th March 2025
Updated Code Received	21st March 2025
Review 2	24th March 2025 - 25th March 2025
Fixed In	02f41fb0c9882358905d49c7a31b9201184bf1fd

Number of Issues per Severity



High	0 (0.00%)
Medium	1 (33.33%)
Low	2 (66.67%)
Informational	0 (0.00%)

Severity

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

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] 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 behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

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 Issues

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

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

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.

Medium Severity Issues

Missing Transfer Success Check in BatchTransfer.sol

Acknowledged

Path

BatchTransfer.sol

Function

BatchTransfer

Description

The BatchTransfer contract facilitates bulk token transfers using the transferFrom function of an IFireFund token contract. However, the implementation fails to properly check whether the transferFrom operation succeeds. The contract relies on a low-level call to execute transfers but does not enforce the return value, potentially leading to failed transfers that are not reverted or handled correctly. If transferFrom fails due to insufficient allowance or balance issues, the transaction may proceed without reverting.

Recommendation

Modify the batchTransfer function to explicitly check and enforce transfer success.

EmberGuard's Team Comment:

We want to maximize the number of successful transfers within a single transaction. Reverting on a single failure would discard all successful transfers, which could inconvenience users who prefer partial execution over none at all.

Low Severity Issues

Contract ownership can transferred to the wrong owner.

Resolved

Path

FireFund.sol

Function

transferOwnership()

Description

Access control is a critical part of smart contract security. Within this contract, access control is handled by the Ownable library which allows the owner to relinquish their rights to a new owner in one function call, transferOwnership(). This is risky if the new owner address passed in is malicious or is a victim of an address poisoning attack where some bytes of the address are changed when copied to the clipboard.

Recommendation

Implement two-step ownership that includes confirmation of ownership transfer from the new owner before it is accepted.

Ownership Renouncement Can Break Critical Functions

Resolved

Path

FireFund.sol

Function

renounceOwnership()

Description

The Ownable contract includes a renounceOwnership() function that allows the contract owner to relinquish ownership. If this function is called on the FireFund contract, it will leave the contract without an owner, potentially disrupting critical functions that rely on the onlyOwner modifier.

Recommendation

Override the renounceOwnership() function to disable its functionality.

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 EmberGuard. We performed our audit according to the procedure described above.

Some issues of Medium and Low severity were found,Which emberguard team Resolved and Acknowledged.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

March, 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com