Safe Edges

Audit Report
October, 2024

Table of Content

Executive Summary

Project Name	Fuelup.fun
Project URL	https://fuelup.fun/
Overview	Fuel Up is a meme coin launchpad and marketplace for fair buying, selling, and trade of meme coins. Assets Fuel Up Network is 100% safe and secure with no pre-mint sale or team allocation to prevent rug pulls. Get started:
Audit Scope	https://github.com/LYNC-WORLD/fuelup-fun-smart-contracts
Language	Sway
Blockchain	Fuel
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	18th oct 2024 - 25th oct 2024
Updated Code Received	18st oct 2024
Second Review	25th oct 2024

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	3	2	1
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	**0**

Checked Vulnerabilities

- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- Re-entrancy
- Malicious libraries
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC’s conformance
- Missing Zero Address Validation
- Revert/require functions
- Upgradeable safety
- Using inline assembly
- Style guide violation
- Parallel Execution safety
- UTXO Model Verification
- FuelVM Opcodes
- Cross-Chain Interactions
- Modular Design
- Access Control Vulnerabilities
- Denial of Service (DoS)
- Oracle Manipulation
- Signature Replay Attacks
- Improper Handling of External Calls
- Proxy Storage Collision
- Use of Deprecated Functions

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 statistic 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

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.

Finding Details

F-1. Asset Reserves Update Mechanism

SEVERITY: Medium

REMEDIATION: See DESCRIPTION
STATUS: fixed, commits

DESCRIPTION:

The **remain_asset_reserves** function lacks a proper update mechanism. Once the contract is live, it cannot adjust or update funds, which may lead to significant calculation errors and operational impacts.
Impact: Inaccurate reserve tracking over time could compromise the integrity of the protocol.

```
        remain_asset_reserves: 0
    }
```

Recommendation:

- Implement a proper update function for reserves.
- Add validation checks for reserve updates.

F-2. Uncapped Fee Percentage

SEVERITY: Medium

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

The fee percentage only checks for values less than or equal to 100 but lacks a reasonable upper limit. This could potentially allow the system to charge users a 100% fee, resulting in users being unable to withdraw funds. The absence of a defined upper limit may lead to centralization and power concentration among those who can afford high fees, creating additional security risks. It is crucial for the system to properly define an upper limit on fees to protect user funds and maintain trust in the contract.

```
    require(cfg.fee_percent <= 100, "fee-percent-cannot-exceed-100");
    storage.cfg.write(cfg);
    initialize_ownership(owner);
```

Impact:

The protocol can charge 100% of user funds as fees.

Recommendation:

require(cfg.fee_percent <= 30, "Fee percentage cannot exceed 30");

Add Additional checks

F-3 Missing Reentrancy Protection

SEVERITY: Medium

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

There is a lack of reentrancy guards in functions that perform external calls. The code does not consistently follow the Checks-Effects-Interactions (CEI) pattern.

Impact

this vulnerability could lead to reentrancy attacks, allowing attackers to drain funds from the contract.

Recommendation:

- **Implement a reentrancy guard modifier.**
- **Follow the CEI pattern strictly.**

F-4 Fee Amount Validation

SEVERITY: Low

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

There is no validation for zero fee amounts before transfer.

```
let fee_amount = (eth_amount * fee_percent) / 100;
transfer(fee_receiver, AssetId::base(), fee_amount);
transfer(msg_sender().unwrap(), asset_id, amount_out);
```

Impact

The code may attempt to transfer zero fees to the receiver, which could be a logical error.

Recommendation:

if fee_amount > 0 {

    // Proceed with fee transfer

}

F-5 Missing Zero Address Check in Withdraw Function

SEVERITY: Low

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

The withdraw function does not validate the recipient address, potentially allowing transfers to the zero address.

```
}

#[storage(read, write)]
fn withdraw_assets_from_pool(pool_asset: AssetId, to: Identity){
```

Impact

This could result in permanent loss of funds if assets are transferred to the zero address..
.

Recommendation:

fn withdraw(to: Address, amount: u64) {

    require(to != Address::from(zero), "Cannot withdraw to zero address");

    // Rest of the function

}

F-6 Unused Timestamp Import

SEVERITY: Informative

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

There is an unused import of block::timestamp.
While there is no security impact, this affects code cleanliness and readability.

H

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

Some issues of informational severity were found,which the Griffy Team has Fixed.

Disclaimer

SafeEdges Smart contract security audit provides services to help identify and mitigate potential security risks in Griffy. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. SafeEdges 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.

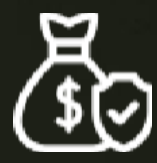
Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Griffy Safe. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay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

SafeEdges 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 Griffy to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About SAFE EDGES

SafeEdges is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

500+
Audits Completed



$3B
Secured

600k+
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2024

https://safeedges.in
info@safeedges.in