QuillAudits

Audit Report
June, 2024

For

FooDriver

Table of Content

Table of Content

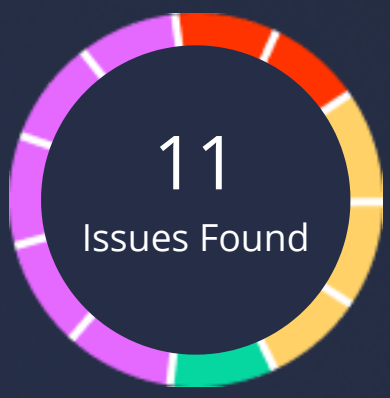
Executive Summary

Project Name	FooDriver
Project URL	https://foodriver.site/
Overview	The FooDriver smart contract system is a suite of interconnected contracts each serving distinct roles within the platform: • FooDriverToken • FooDriverRegistry • FooDriverFactory • FooDriverStore • FooDriverBank Each contract plays a vital role in the ecosystem, from managing financial transactions to handling user roles and permissions. The Foodriver Coin is a utility token designed to facilitate platform operations and smart contract execution. It does not confer ownership rights or profit-sharing, distinguishing it from a security token. Its primary function is to ensure efficient use of the platform's features while adhering to regulatory guidelines.
Timeline	3rd June 2024 - 11th June 2024
Update Code Received	24th June 2024
Second Review	24th June 2024
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 analyse the FooDriver codebase for quality, security, and correctness.
Source Code	https://github.com/mobile-foodriver-systems/smart-contract

Executive Summary

Commit Hash	943cc21f8edfa45d7d63e739024141c3ae61fb49
Contracts In-Scope	- contracts/FooDriverBank.sol - contracts/FooDriverStore.sol - contracts/FooDriverFactory.sol - contracts/FooDriverRegistry.sol - contracts/FooDriverToken.sol - contracs/TokenLock.sol
Branch	Main
Contracts Out of Scope	In-scope contract has been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities. Below are Out of Scope Contracts: - OpenZeppelin contracts (Initializable.sol, ...)
Fixed In	04640db72a406211b9213bb6acb469747a3b03ef

Number of Security Issues per Severity



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	2	0	4

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
- 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 behavior.
- Token distribution and calculations are as per the intended behavior 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

Manual Review, Foundry, Slither.

Types of Severity

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

High Severity Issues

A high severity issue or vulnerability means 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 impacts 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 issues were identified in the initial audit and 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.

High Severity Issues

1. Missing token transfer validation could make functionality break due to improperly handled transfers

Path

FooDriverToken.sol, FooDriverBank.sol, FooDriverStore.sol

Function

purchasePublic, purchasePrivate, releasePayment, refundPayment, withdrawERC20Token

Description

Tokens that comply with the ERC20 standard are expected to:

- Moves amount tokens from the caller's account to recipient.
- Returns a boolean value indicating whether the operation succeeded.
- Emits a Transfer event.

After an ERC20 token transfer, it is expected that the 3 conditions above are satisfied. It is recommended to validate the transfers with a check on the boolean value returned as some tokens do not fully comply with this specification (commonly called weird-erc20-tokens). If there is no validation on the return value of the transfer or transferFrom calls, the remainder of the function could run to completion without the tokens being transferred.

The purchasePublic and purchasePrivate functions call transferFrom on the token address passed and are vulnerable to this attack - if the function calls fail or revert for any reason (such as insufficient token balances) tokens will still be minted. Users will be able to effectively game the protocol by minting tokens in either of these sale periods without paying out those tokens.

Some other issues could arise are with:

- Fee-on-transfer tokens where an incomplete amount of tokens is sent into the contract with the same rate calculations as regular transfers would.
- Tokens with irregular token decimals (less or greater than 18 decimals) being used in the contracts would affect the token accounting as well because a user could pay for the private or public sale with a less valuable token that has an inflated amount of decimals making it appear to be worth more.

FooDriverBank also doesn't have a check on the return values in releasePayment and refundPayment, this would silently revert and leave tokens stuck or lost instead of being transferred to the customer, store, courier or commission wallet.

Recommendation

- Use the SafeERC20 token wrapper from OpenZeppelin to catch possible weird ERC20 transfer nuances.
- Since the supportedTokens allowlist is used, tokens to be added should be properly vetted before being used in the system.

Status

Resolved

2. Incorrect rate accounting would give users 100x more tokens than intended at mint

Path

FooDriverToken.sol

Function

updateTokenRates, purchasePrivate, purchasePublic

Description

The rates for the private and public sale, privateSaleRate and publicSaleRate describe a 100 to 1 rate to support floating numbers. When these values get set in the updateTokenRates function it allows the following require checks to pass
require(supportedTokens[token].privateSaleRate > 0, “Unsupported token or rate is 0”);
require(supportedTokens[token].publicSaleRate > 0, “Unsupported token or rate is 0”);

However it doesn’t make the payout calculations correct.

Provided this is a scenario where:
1% is intended to be represented as 100 to support floating point numbers, then an issue would arise with representing 100% as 10,000 because payout would be 1,000e18 * 10000/100 = 100,000e18 tokens. This would give users 100x more tokens than were initially intended on a single purchase thereby depleting the tokensale balance sooner than expected and the cost of 1x.

and not the supportedToken to token price (assuming USDC is the supported token) i.e. 100 USDC = 1 FDT, 1 USDC = 0.01 FDT

Recommendation

- Consider using a basis points system where divisions for percentages are done by 10,000 instead of 100.
- Properly document the intention with rates beyond the comment provided.

Status

Resolved

Medium Severity Issues

3. Inaccurate token pricing in FooDriverToken could allow users get more tokens at less cost to them

Path

FooDriverToken.sol

Function

supportedTokens

Description

The FooDriverToken contract allows users to purchase tokens in the private sale and the public sale via supportedTokens added by the registry. The issue here lies with the rates which are manually added - if there happens to be a change in the price of the tokens supported via a depeg or improper rate adjustments before the user with DEFAULT_ADMIN_ROLE gets to update it, tokens will be improperly priced until that update occurs.

Recommendation

The team can consider using oracles to get prices and adjust the token sale rates automatically.

Status

Acknowledged

4. Missing gap variables could cause storage collisions when upgrades occur

Path

FooDriverToken.sol, FooDriverFactory.sol, FooDriverRegistry.sol, FooDriverBank.sol

Description

Upgradeable contracts allow for smart contracts to have extended functionality. They can be deployed as is, and upgraded at a later time to adjust the implementation. Within composable upgradeable contracts, the issue of storage collision exists which could alter the reading of storage for each of the contracts built atop each other.

To avoid issues when reading or writing to storage it is advisable to have a gap introduced in each contract to prevent the new storage variables added from crossing into existing storage locations. Here, OpenZeppelin describes in detail how to manage smart contract storage with gaps:

Writing Upgradeable Contracts - OpenZeppelin Docs

Recommendation

Include storage gaps at the end of state variable declarations in each of the contracts affected.

Status

Resolved

5. Missing input validation for varying percentage splits on orders created

Path

FooDriverStore.sol

Function

createOrder()

Description

For orders created by the Store contract via the registry, there is no input validation for the different percentages that are passed in as arguments. It is expected that these values should not exceed 100% of the total amount but not enforced in the codebase. This would make some functions to revert or continue to run till the end and fail to transfer the expected token balances to the respective parties.

Recommendation

Ensure that the balance to be distributed falls within the expected ranges by ensuring all token amounts do not overflow 100%. The invariant for tokensLocked should be greater or equal to tokensReleased.

Status

Resolved

Low Severity Issues

6. Low test coverage

Description

Unit tests are used to ensure that the code functions as expected by passing in varying user input and setting up various parameters to test the boundaries of the protocol. There are no unit test cases associated with the codebase provided, hereby increasing the probability of bugs being present and reducing quality assurance.

Recommendation

Include unit tests that have > 95% code coverage including all possible paths for code execution.

Status

Acknowledged

Informational Issues

7. No duration for public and private sale could make registry arbitrarily start the sale, mint all the tokens and end the sale

Path

FooDriver.sol

Description

The current implementation of the token sale does not have any timing schedules for users to work with. Some of the issues with this are:

- The public and private sale could happen concurrently, and this would cause some concern because public sales usually happen after the private sale is ended or fully
- subscribed to.
  There is also no duration given for the tokens to be bought within, both sales could happen in a single hour not allowing the community to fully participate.

Recommendation

- purchasePublic could include a check for the private sale to have ended before the public sale can begin.
- Include a duration for the tokensale in the contract.

Status

Resolved

8. FooDriverToken can mint 0 tokens in endPublicSale and endPrivateSale

Path

FooDriverToken.sol

Function

endPublicSale, endPrivateSale

Description

The ERC20 internal _mint function does not revert when 0 tokens are passed in as an argument. A transfer event will also be emitted for the 0 token mints which would be unnecessary, especially for on-chain monitoring software.

Recommendation

- endPublicSale and endPrivateSale could check that the amount left is greater than 0 before calling mint.
- Include a duration for the tokensale in the contract.

Status

Resolved

9. Code reuse

Description

The ERC20 internal _mint function does not revert when 0 tokens are passed in as an argument. A transfer event will also be emitted for the 0 token mints which would be unnecessary, especially for on-chain monitoring software.

Recommendation

FooDriverHelper.sol could be created to hold all repeated code blocks, shorten the lines of code and possibly reduce deployment cost generally.

Status

Acknowledged

10. Events can be emitted out of order

Path

FooDriverToken.sol, FooDriverBank.sol, FooDriverFactory.sol

Description

The Check-Effects-Interact [CEI] pattern helps to greatly reduce the risk of reentrancy in contracts, it is a best practice to emit events as part of the Effects before any external calls or interactions happen which could open up a reentrancy vulnerability.

Recommendation

Emit events before external calls or prior to updating the state of the contract.

Status

Resolved

11. Wrong comments

Path

FooDriverRegistry.sol

Description

The comment * @notice Retrieves the address of a store by its ID. does not describe accurately what the function returns. The return value is a specific interface for the FooDriverStore, not an address as the comment says.

Recommendation

To avoid issues with future integrations, properly label comments.

Status

Resolved

General Requirements

- Lock the solidity pragma version before deployment
- Variables only set once (tokenSupply, tokenForPresale) can be made constant or immutable

Functional Tests Cases

- ✓ Owner can withdraw tokens
- ✓ Owner can end presale to allow users begin claims
- ✓ Create order and lock funds
- ✓ Release and Refund cases check
- ✓ Able to change courier address
- ✓ All users can withdraw presale tokens

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 FooDriver codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, 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 FooDriver smart contract. 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 FooDriver smart contract. 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 FooDriver 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.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

FoodDriver

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com