farmsent

Token

SMART CONTRACT AUDIT REPORT

IMMUNEBYTES

31 March 2023

Introduction

About Farmsent

Farmsent is a groundbreaking platform that is revolutionizing the food trade and promoting food security. It is the world's first blockchain technology platform designed specifically for farmers, providing solutions to the food industry while prioritizing the interests of farmers

The platform creates a decentralized marketplace for agricultural products such as rice, coffee, and wheat, connecting farmers with buyers and businesses. This benefits smaller farmers who may not have the resources to compete with larger players in the industry, giving them a fair chance to sell their products to a global market.

Visit https://www.farmsent.io/ to know more about it.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The team has provided the following doc for audit:

1. https://www.farmsent.io/assets/images/Farmsent-whitepaper.pdf

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safely used third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

A team of independent auditors audited the code, including -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	Farment
Platform	EVM
Languages	Solidity
GitHub Link	https://github.com/Blockchainxtech/farmsent-token
Commit - Final Audit	7ee0c98e2ea861174a78f8dc397cad2ad6d24559
Platforms & Tools	Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz

Security Level References

Every issue in this report was assigned a severity level from the following:

CRITICAL

Issues may result in fund leakage or incorrect fund allocation.

HIGH

Issues affecting the business logic, performance, and functionality.

MEDIUM

Issues could lead to data loss or other manipulations.

LOW

Issues around smart contract code upgradability, libraries, and others.

INFORMATIONAL

Issues which can further improve the code on gas optimizations and reusability.

Issues	Critical	High	Medium	Low	Informational
Open	-	-	-	-	-
Closed	-	-	1	4	2
Acknowledged	-	-	-	-	-

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Finding

#	Findings	Risk	Status
1.	initializer() modifier should be removed from constructor	**Medium**	**Fixed**
2.	Absence of Input Validations in intialize() function	**Low**	**Fixed**
3.	Redundant functions increase contract size unnecessarily	**Low**	**Fixed**
4.	Invalid error messages found	**Low**	**Fixed**
5.	External Visibility could be used	**Low**	**Fixed**
6.	Coding Style Issues in the Contract	**Informational**	**Fixed**
7.	*initialize() function has an invalid error message*	**Informational**	**Fixed**

Critical Severity Issues

No issues were found.

High Severity Issues

No issues were found.

Medium severity issues

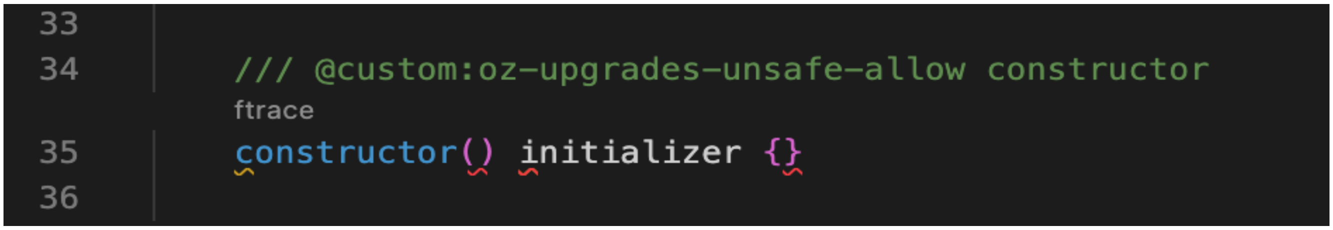
1. **initializer() modifier should be removed from constructor**

Line no: 35

Description: The Farmsent contract is an upgradeable contract where the initialize() function acts as the constructor.

The initializer modifier basically ensures that the initialize() function can only be called once. For this to work, the initializer modifier must only be executed once, i.e., before the execution of the initialize() function.

However, during the manual review, it was found that the initializer modifier has also been assigned to the constructor of the contract which technically will use this modifier before the initialize() function. This will lead to an unwanted scenario where the deployer won't be able to initialize() the contract at all after deployment.



```
33
34      /// @custom:oz-upgrades-unsafe-allow constructor
        ftrace
35      constructor() initializer {}
36
```

Recommendation: The initializer modifier shall only execute once in the initialize() function and therefore must be removed from the constructor.

Status: Amended
The team has fixed the issue.

Low severity issues

1. Absence of Input Validations in intialize() function

Line no: 37-50

Description: The initialize() function allows passing imperative arguments like decimal for the token and the total supply.

However, during the review, it was found that no adequate input validations have been provided in the functions.

Although this function acts as a constructor, validating every argument being passed in functions is still recommended.

Recommendation: Include require statements to ensure only valid arguments are passed to functions.

Status: Amended
The team has fixed the issue.

2. Redundant functions increase contract size unnecessarily

Line no: 208-213, 256-260

Description: The contract includes additional private view functions like checkWhitelisted() and checkBlacklisted() to simply return whether or not an address is whitelisted.

However, these functions can easily be removed if the _whitelist and _blacklist mappings are made public.

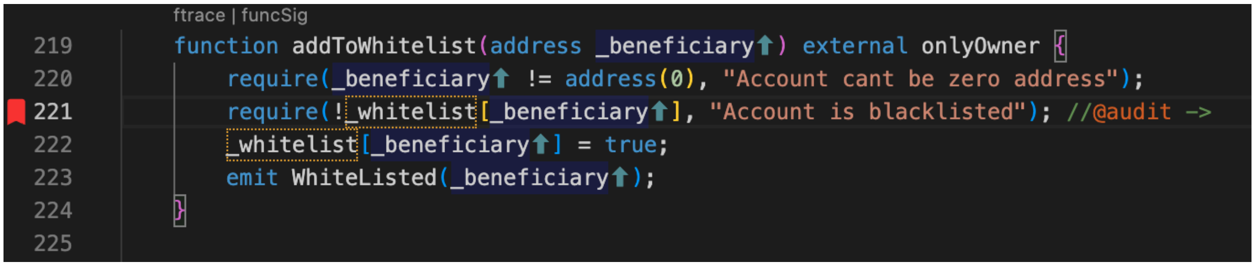
Recommendation: Adding redundant functions or state variables eventually increases the contract bytecode unnecessarily. Code can be optimized.

Status: Amended
The team has fixed the issue.

3. Invalid error messages found

Line no: 221

Description:
The require statement in the addToWhitelist() function at the above-mentioned line includes an invalid error message.



```
          ftrace | funcSig
219       function addToWhitelist(address _beneficiary) external onlyOwner {
220           require(_beneficiary != address(0), "Account cant be zero address");
221           require(!_whitelist[_beneficiary], "Account is blacklisted"); //@audit ->
222           _whitelist[_beneficiary] = true;
223           emit WhiteListed(_beneficiary);
224       }
225
```

Since it ensures that the address argument shouldn't already be whitelisted, the error message should be replaced with - "Address is already whitelisted".

Recommendation: Invalid error messages lead to ambiguity. Error messages should be accurate.

Status: Amended
The team has fixed the issue.

4. External Visibility could be used

Description: Functions that are never called within the contract should be marked as external visibility instead of public visibility.

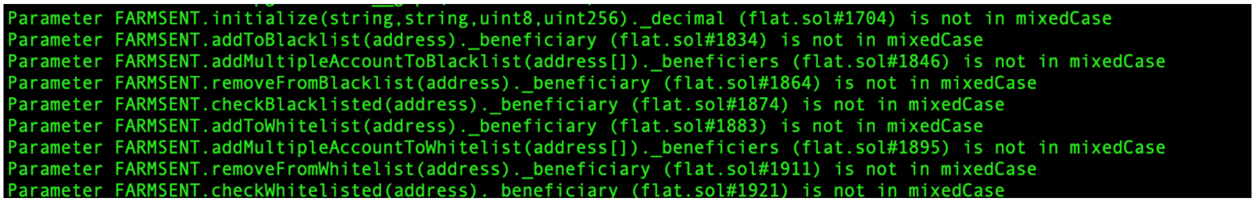
Recommendation: If public visibility of such functions isn't intentional in the contract, they can be marked as external.

Status: Amended
The team has fixed the issue.

Informational

1. Coding Style Issues in the Contract

Description: The code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios, may lead to bugs in the future.



```
Parameter FARMSENT.initialize(string,string,uint8,uint256)._decimal (flat.sol#1704) is not in mixedCase
Parameter FARMSENT.addToBlacklist(address)._beneficiary (flat.sol#1834) is not in mixedCase
Parameter FARMSENT.addMultipleAccountToBlacklist(address[])._beneficiers (flat.sol#1846) is not in mixedCase
Parameter FARMSENT.removeFromBlacklist(address)._beneficiary (flat.sol#1864) is not in mixedCase
Parameter FARMSENT.checkBlacklisted(address)._beneficiary (flat.sol#1874) is not in mixedCase
Parameter FARMSENT.addToWhitelist(address)._beneficiary (flat.sol#1883) is not in mixedCase
Parameter FARMSENT.addMultipleAccountToWhitelist(address[])._beneficiers (flat.sol#1895) is not in mixedCase
Parameter FARMSENT.removeFromWhitelist(address)._beneficiary (flat.sol#1911) is not in mixedCase
Parameter FARMSENT.checkWhitelisted(address)._beneficiary (flat.sol#1921) is not in mixedCase
```

During the automated testing, it was found that the Farmsent contract had quite a few code-style issues. Please follow this link to find details on naming conventions in solidity code.

Recommendation: Therefore, it is recommended to fix issues like naming convention, indentation, and code layout issues in a smart contract.

Status: Amended
The team has fixed the issue.

2. initialize() function has an invalid error message

Line no - 42

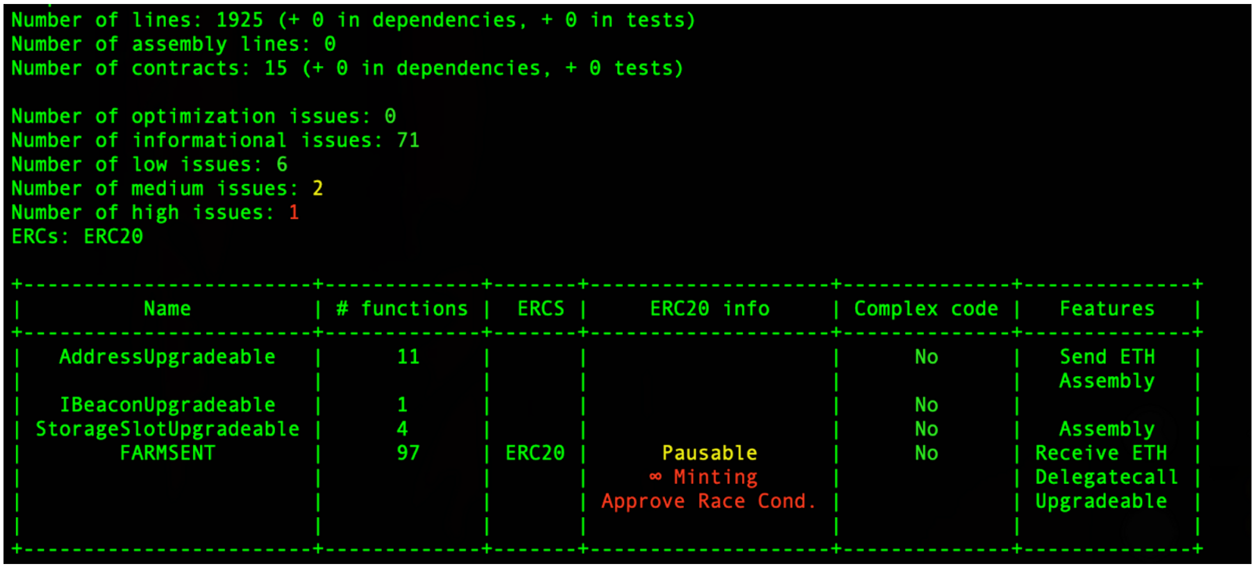
Description: The error message at Line 42 indicates that the decimal value cannot be equal to 18.

This is invalid because the **require statement condition** allows passing 18 as the decimal value for the token.

Recommendation: Error messages should be accurate.

Status: Amended
The team has fixed the issue.

Automated Test Results



```
Number of lines: 1925 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 15 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 71
Number of low issues: 6
Number of medium issues: 2
Number of high issues: 1
ERCs: ERC20

+------------------------+-------------+-------+--------------------+--------------+--------------+
|          Name          | # functions |  ERCS |     ERC20 info     | Complex code |   Features   |
+------------------------+-------------+-------+--------------------+--------------+--------------+
|   AddressUpgradeable   |      11     |       |                    |      No      |   Send ETH   |
|                        |             |       |                    |              |   Assembly   |
|   IBeaconUpgradeable   |      1      |       |                    |      No      |              |
| StorageSlotUpgradeable |      4      |       |                    |      No      |   Assembly   |
|        FARMSENT        |      97     | ERC20 |      Pausable      |      No      | Receive ETH  |
|                        |             |       |     ∞ Minting      |              | Delegatecall |
|                        |             |       | Approve Race Cond. |              | Upgradeable  |
|                        |             |       |                    |              |              |
+------------------------+-------------+-------+--------------------+--------------+--------------+
```

Concluding Remarks

While conducting the audits of the Farmsent, it was observed that the contracts contain Medium and Low severity issues along with a few recommendations.

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Farmsent platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

IMMUNEBYTES

AUDITS