Main Street

Smart Contract Audit Report

July 30, 2021

IMMUNE BYTES

Introduction

1. About Main Street

Main Street ($MAINst) is a deflationary token offered on the Binance Smart Chain. The goal of Main Street is to answer the question so often asked, "Why should we hold?". Main Street provides its holders the opportunity to find new tokens in our Neighborhood and Alley, featuring tokens ranging from high use case, to newly listed and unvetted. Combined with the addition of our Main Street shops which will provide entertainment and games, we have created a hub that brings reason and fun to trading cryptocurrency.

Visit https://www.buymainstreet.com/ to know more about.

2. About ImmuneBytes

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

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

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

Documentation Details

The Main Street team has provided the following doc for the purpose of audit:

1. Audit document.docx

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 safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any 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 in order 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.

The code was audited by a team of independent auditors which includes -

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: Main Street
- Contracts Name: BuyMainStreet
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: 0x8fc1a944c149762b6b578a06c0de2abd6b7d2b89
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck, Fuzz

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 within 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

Security Level References

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

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	2	4
Closed	-	-	-

High severity issues

No issues found.

Medium severity issues

1. **_getTaxFee function does not hold any significance.**
   Line no - 830-832

   Description:
   The protocol includes a function called **_getTaxFee** that returns the **_TAX_FEE** state variable in the protocol.

```
829
830  function _getTaxFee() private view returns(uint256) {
831      return _TAX_FEE;
832  }
833
```

   However, the function has been marked as private but never called from within the contract at any instance. Moreover, since the visibility specifier is **private** the function cannot be called from outside the contract as well.

   Recommendation:
   If the **getTaxFee** doesn't hold any significance in the protocol, it should be removed to save space and gas.
   However, if it's a necessary function, the Function should either be marked as External or be called from within the contract at some instance to ensure that it is used adequately.

2. **Loops are extremely costly**
   Line no -615, 795

   Description:
   The **BuyMainStreetToken** contract has some **for loops** in the contract that include state variables like .length of a non-memory array, in the condition of the for loops.

   As a result, these state variables consume a lot more extra gas for every iteration of the for loop. The following function includes such loops at the above-mentioned lines:
   - **includeAccount**
   - **_getCurrentSupply**

```
613     function includeAccount(address account) external onlyOwner() {
614         require(_isExcluded[account], "Account is already included");
615         for (uint256 i = 0; i < _excluded.length; i++) {
616             if (_excluded[i] == account) {
617                 _excluded[i] = _excluded[_excluded.length - 1];
```

Recommendation:
It's quite effective to use a local variable instead of a state variable like .length in a loop. This will be a significant step in optimizing gas usage.

For instance,

```
function includeAccount(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already included");

     uint256 local_variable = _excluded.length; // Storing Length in a local Variable
    for (uint256 i = 0; i < local_variable; i++) {
       if (_excluded[i] == account) {
          _excluded[i] = _excluded[_excluded.length - 1];
          _tOwned[account] = 0;
          _isExcluded[account] = false;
          _excluded.pop();
          break;
       }
       }
       }
```

Low severity issues

1. **Absence of Error messages in Require Statements**
   Line no - 632

   Description:
   The **BuyMainStreetToken** contract includes a **require statement** in functions(at the above-mentioned lines) that doesn't contain any error message.

```
631     function updateFee(uint256 _txFee,uint256 _burnFee,uint256 _marketingFee) onlyOwner() public{
632         require(_txFee < 100 && _burnFee < 100 && _marketingFee < 100);
633         _TAX_FEE = _txFee* 100;
634         _BURN_FEE = _burnFee * 100;
```

   While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

Recommendation:
Error Messages must be included in every require statement in the contract

2. **Absence of Zero Address Validation**

Description:
During the automated testing, it was found that the contract includes quite a few functions that update an imperative address in the contract like **FeeAddress**.

However, no Zero Address Validation is implemented on the following function while updating such state variables of the contract:

- **setAsmarketingAccount**

Recommendation:
A **require** statement should be included in such functions to ensure no zero address is passed in the arguments.

3. **External Visibility should be preferred**

Description:
Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as **external** within the contract:

- **isExcluded()**
- **totalFees()**
- **deliver()**
- **reflectionFromToken()**
- **totalBurn()**
- **totalmarketing()**
- **updateFee()**

Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

4. **Constant declaration should be preferred**
Line no- 469, 471

Description:
State variables that are not supposed to change throughout the contract should be declared as **constant**.

Recommendation:
The following state variables need to be declared as **constant**, unless the current contract design is intended.

- **_GRANULARITY**
- **_MAX**

Recommendations

1. **Coding Style Issues in the Contract**

Description:
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.

```
Variable BuyMainStreetToken._NAME (contracts/buyMainStreet.sol#464) is not in mixedCase
Variable BuyMainStreetToken._SYMBOL (contracts/buyMainStreet.sol#465) is not in mixedCase
Variable BuyMainStreetToken._DECIMALS (contracts/buyMainStreet.sol#466) is not in mixedCase
Variable BuyMainStreetToken.FeeAddress (contracts/buyMainStreet.sol#467) is not in mixedCase
Variable BuyMainStreetToken._MAX (contracts/buyMainStreet.sol#469) is not in mixedCase
Variable BuyMainStreetToken._DECIMALFACTOR (contracts/buyMainStreet.sol#470) is not in mixedCase
Variable BuyMainStreetToken._GRANULARITY (contracts/buyMainStreet.sol#471) is not in mixedCase
Variable BuyMainStreetToken._TAX_FEE (contracts/buyMainStreet.sol#480) is not in mixedCase
Variable BuyMainStreetToken._BURN_FEE (contracts/buyMainStreet.sol#481) is not in mixedCase
Variable BuyMainStreetToken._marketing_FEE (contracts/buyMainStreet.sol#482) is not in mixedCase
Variable BuyMainStreetToken.ORIG_TAX_FEE (contracts/buyMainStreet.sol#485) is not in mixedCase
Variable BuyMainStreetToken.ORIG_BURN_FEE (contracts/buyMainStreet.sol#486) is not in mixedCase
Variable BuyMainStreetToken.ORIG_marketing_FEE (contracts/buyMainStreet.sol#487) is not in mixedCase
```

During the automated testing, it was found that the **BuyMainStreetToken** contract had quite a few code style issues.

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

2. **NatSpec Annotations must be included**

Description:
The smart contracts do not include the NatSpec annotations adequately.

Recommendation:
Cover by NatSpec all Contract methods.

Automated Audit Result

```
Compiled with solc
Number of lines: 835 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 23
Number of informational issues: 112
Number of low issues: 5
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+-----------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code | Features  |
+--------------------+-------------+-------+--------------------+--------------+-----------+
|      SafeMath      |      8      |       |                    |      No      |           |
|      Address       |      7      |       |                    |      No      | Send ETH  |
|                    |             |       |                    |              | Assembly  |
| BuyMainStreetToken |      57     | ERC20 |    No Minting      |     Yes      |           |
|                    |             |       | Approve Race Cond. |              |           |
|                    |             |       |                    |              |           |
+--------------------+-------------+-------+--------------------+--------------+-----------+
INFO:Slither:contracts/buyMainStreet.sol analyzed (6 contracts)
```

Concluding Remarks

While conducting the audits of the Main Street smart contract, it was observed that the contracts contain Medium and Low severity issues along with a few areas of 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 Main Street 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 the code refactor by the team on critical issues.

ImmuneBytes