Animal Token

Stray

Smart Contract Audit Report

August 16, 2021

Introduction

1. About Animal Token

Animal Token is a project designed to help homeless animals. The main goal of the Animal Token project is to provide all possible assistance to homeless animals, such as:

- charitable activities to existing shelters
- organization of animal shelters around the world
- assistance in adoption

A separate fund is being created for assistance, to which 0.3% of each transaction will be deducted. Token holders will also be able to participate in voting related to the choice of certain measures to promote shelters and help homeless animals.

A community of people united by the idea of helping homeless animals, developing shelters and adoption will be gathered around the project.

Visit https://animaltoken.one/ to learn 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 65+ 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 Animal Token team has provided the following doc for the purpose of audit:

1. https://animaltoken.one/test/animaltoken_wp.pdf

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: Animal Token
- Contracts Name: Stray
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: 6303346cfa62d5002ab055196d220e7c15faf0cc
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

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. **Multiplication is being performed on the result of Division**
   Line no - 208-210, 212-214, 220-222

   Description
   During the manual code review and automated testing of the **Stray** contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division.

   Integer Divisions in Solidity might be truncated. Moreover, this performing division before multiplication might lead to loss of precision.

   The following functions involve division before multiplication in the mentioned lines:
   - _transfer at 208-210, 212-214, 220-222

```
207         if (_isExcluded[sender] && !_isExcluded[recipient]) {
208             _transferFromExcluded(sender, recipient, amount.div(1000).mul(995));
209             _transferFromExcluded(sender, _charity, amount.div(1000).mul(3)); // _charity
210             _transferFromExcluded(sender, _team, amount.div(1000).mul(2)); // team
211         } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
212             _transferToExcluded(sender, recipient, amount.div(1000).mul(995));
213             _transferStandard(sender, _charity, amount.div(1000).mul(3)); // _charity
214             _transferStandard(sender, _team, amount.div(1000).mul(2)); // team
215         } else if (_isExcluded[sender] && _isExcluded[recipient]) {
216             _transferBothExcluded(sender, recipient, amount.div(1000).mul(995));
217             _transferFromExcluded(sender, _charity, amount.div(1000).mul(3)); // _charity
218             _transferFromExcluded(sender, _team, amount.div(1000).mul(2)); // team
219         } else {
220             _transferStandard(sender, recipient, amount.div(1000).mul(995));
221             _transferStandard(sender, _charity, amount.div(1000).mul(3)); // _charity
222             _transferStandard(sender, _team, amount.div(1000).mul(2)); // team
223         }
```

Automated Test Results

```
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferFromExcluded(sender,recipient,amount.div(1000).mul(995)) (FlatStray.sol#810)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferToExcluded(sender,recipient,amount.div(1000).mul(995)) (FlatStray.sol#814)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferStandard(sender,_charity,amount.div(1000).mul(3)) (FlatStray.sol#815)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferBothExcluded(sender,recipient,amount.div(1000).mul(995)) (FlatStray.sol#818)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferFromExcluded(sender,_charity,amount.div(1000).mul(3)) (FlatStray.sol#811)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferStandard(sender,_team,amount.div(1000).mul(2)) (FlatStray.sol#816)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferStandard(sender,recipient,amount.div(1000).mul(995)) (FlatStray.sol#822)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferFromExcluded(sender,_charity,amount.div(1000).mul(3)) (FlatStray.sol#819)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferFromExcluded(sender,_team,amount.div(1000).mul(2)) (FlatStray.sol#812)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferStandard(sender,_charity,amount.div(1000).mul(3)) (FlatStray.sol#823)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferStandard(sender,_team,amount.div(1000).mul(2)) (FlatStray.sol#824)
STRAY._transfer(address,address,uint256) (FlatStray.sol#805-826) performs a multiplication on the result of a division:
        -_transferFromExcluded(sender,_team,amount.div(1000).mul(2)) (FlatStray.sol#820)
```

Recommendation

Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if they do not lead to expected results.

2. **Costly Loops found in the Protocol**
   Line no - 184, 295

Description

The **Stray** 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**

```
182     function includeAccount(address account) external onlyOwner() {
183         require(_isExcluded[account], "Account is already excluded");
184         for (uint256 i = 0; i < _excluded.length; i++) {
185             if (_excluded[i] == account) {
186                 _excluded[i] = _excluded[_excluded.length - 1];
187                 _tOwned[account] = 0;
188                 _isExcluded[account] = false;
189                 _excluded.pop();
190                 break;
191             }
```

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 Zero Address Validation**
   Line no- 131-145

   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 **_rewardDistributor, _marketing** etc .

   However, no Zero Address Validation is implemented on the following function while updating such state variables of the contract:
   - **setRewardDistributorAccount**
   - **setMarketingAccount**
   - **setCharityAccount**
   - **setTeamAccount**

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

2. **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:
   - **reflectionFromToken()**
   - **totalFees()**
   - **reflect()**
   - **isExcluded()**

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

3. **Constant declaration should be preferred**
   Line no- 24, 25, 26, 33, 35

   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.
   - **_burnAddress**
   - **_decimals**
   - **_monthlyDistribution**
   - **_name**
   - **_symbol**

4. **Redundant State Variable Update**
   Line no: 36

   Description
   The **Stray** Smart contract involves the redundant updating of a State variable in the contract at the above-mentioned line

```
36      uint256 private _distributedMonths = 0;
37
```

   A boolean variable is by-default initialized to FALSE whereas a uint256 is initialized to ZERO. Hence, such state variables do not need to be initialized explicitly.

   Recommendation
   Redundant initialization of state variables should be avoided.

Recommendations

1. **Contract includes Hardcoded Addresses**
   Line no - 33

   Description
   Keeping in mind the immutable nature of smart contracts, it is not considered a better practise to hardcode any address in the contract before deployment.

```
32
33      address private _burnAddress = 0x000000000000000000000000000000000000dEaD;
34
```

Recommendation
Instead of including hardcoded addresses in the contract, initialize those addresses within the constructors at the time of deployment.

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: 905 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 22
Number of informational issues: 72
Number of low issues: 6
Number of medium issues: 12
Number of high issues: 0

ERCs: ERC20

+----------+-------------+-------+--------------------+--------------+--------------+
|   Name   | # functions |  ERCS |     ERC20 info     | Complex code |   Features   |
+----------+-------------+-------+--------------------+--------------+--------------+
| SafeMath |      13     |       |                    |      No      |              |
| Address  |      11     |       |                    |      No      |   Send ETH   |
|          |             |       |                    |              | Delegatecall |
|          |             |       |                    |              |   Assembly   |
|  STRAY   |      51     | ERC20 |     No Minting     |      No      |              |
|          |             |       | Approve Race Cond. |              |              |
|          |             |       |                    |              |              |
+----------+-------------+-------+--------------------+--------------+--------------+
INFO:Slither:FlatStray.sol analyzed (6 contracts)
```

Concluding Remarks

While conducting the audits of the Animal Token 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 Animal Token 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 Animal Token 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