Animal Token Smart Contract Final Audit Report

Project Synopsis

Project Name	**Animal Token**
Platform	Ethereum, Solidity
Github Repo	Not Provided
Deployed Contract	Not provided
Total Duration	4 Days
Timeline of Audit	**13th August to 16th August 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	**Animal Token**
Language	Solidity
Commit Hash	**Null**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	1	0
Medium Severity	0	2
Low Severity	2	2
Information	3	0
Total Found	6	4

Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

A. Contract Name: STRAY Token

High Severity Issue

A.1 *_transfer* function includes invalid token transfer for Team address

Line no - 219

Status: OPEN

Explanation:

During the manual code review of the updated contract it was found that the **_transfer** function in the protocol involves a wrong transfer of tokens for the first **IF Statement condition.**

In the first ***IF Statement***(Line 216-220), the tokens being transferred to the **team address** are the same as the tokens being transferred to the **recipient address. (*Line 219*)**

While the **team address** should receive the token amount stored in the local variable ***teamAmount*,** it actually receives the token amount stored in ***_toAmount*** local variable.

This is an invalid logic as the amount of token transferred to the team address will be significantly larger than the actual team amount and this will break the intended behavior of the function.

```
207 ▾    function _transfer(address sender, address recipient, uint256 amount) private {
208          require(sender != address(0), "ERC20: transfer from the zero address");
209          require(recipient != address(0), "ERC20: transfer to the zero address");
210          require(amount > 0, "Transfer amount must be greater than zero");
211
212          uint256 charityAmount = amount.mul(3).div(1000);
213          uint256 teamAmount = amount.mul(2).div(1000);
214          uint256 toAmount = amount.mul(995).div(1000);
215
216 ▾        if (_isExcluded[sender] && !_isExcluded[recipient]) {
217              _transferFromExcluded(sender, recipient, toAmount);
218              _transferFromExcluded(sender, _charity, charityAmount); // _charity
219              _transferFromExcluded(sender, _team, toAmount); // team
```

Recommendation:

The transfer of token procedure should be thoroughly tested with adequate test scripts and the **_transfer** function should be updated adequately to avoid the above-mentioned scenario.

Medium Severity Issues

A.1 Multiplication is being performed on the result of Division

Line no - *208-210, 212-214, 220-222*

Status: CLOSED

***Explanation*:**

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.

A.2 Costly Loops found in the Protocol

Status: CLOSED

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

A.3 Absence of Zero Address Validation

Status: Not Considered

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.

A.4 External Visibility should be preferred

Status: CLOSED

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.

A.5 Constant declaration should be preferred

Status: Not Considered

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***

A.6 Redundant State Variable Update

Status: CLOSED

Line no: 36

Explanation

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.

Informational

A.7 Contract includes Hardcoded Addresses

Line no - 33

Status: Not Considered

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.

A.8 NatSpec Annotations must be included

Status: Not Considered

Description:

The smart contracts do not include the NatSpec annotations adequately.

Recommendation:

Cover by NatSpec all Contract methods.

A.10 Console.log statements found in the Solidity Code

Line no - 671, 672

Description:

During the manual code review of the updated contract it was found that the contract includes some console log statements for debugging or testing purposes.

```
668     function distributeMonthlyReward() external onlyRewardDistributor {
669         require(_distributedMonths < 6, "Can only distribute 6 times");
670         _distributedMonths++;
671         console.log("a");
672         console.log(balanceOf(_msgSender()));
673         _transfer(_msgSender(), address(this), _monthlyDistribution);
674     }
```

Recommendation:

It would be effective if such statements are removed before deploying the code.