FortuneCookieV2 Smart Contract Preliminary Audit Report

Project Synopsis

Project Name	**Fortune Cookie**
Platform	Binance Smart Chain, Solidity
Github Repo	https://bscscan.com/address/0xca94698f5a683939700ea611d6ada30cae632a9d#code
Deployed Contract	Yes
Total Duration	4 Days
Timeline of Audit	**31st May 2021 to 3rd June 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	FortuneCookieV2
Language	Solidity
Commit Hash	**Null**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	3	0
Medium Severity	3	0
Low Severity	4	0
Information	2	0
Total Found	12	0

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: FortuneCookieV2

High Severity Issues

A.1 *_handleLottery* function includes a similar operation twice in its function body

Line - 1159 - 1242

Explanation:

As per the current design of the **_handleLottery** function, it includes a call to the **insertUser** function to add the user details in the contract. It then increments the state variable **_txCounter** with 1.

```
1159    function _handleLottery(address recipient, uint256 p
1160
1161        if (_countUsers == 0 || potContribution == 0) {
1162            // Register the user if needed.
1163            if(isUser(recipient) != true) {
1164                insertUser(recipient, 0);
1165            }
1166
1167            _txCounter += 1;
1168
1169            return true;
1170        }
```

However, an exact similar operation is being performed at the end of the function as well.(Line 1235-1239).

```
1235            if(isUser(recipient) != true) {
1236                insertUser(recipient, 0);
1237            }
1238
1239            _txCounter += 1;
```

While the call to **insertUser** function is being guarded by the if statements, the increment in the **_txCounter(**Line 1239**)** is not. This might lead to an undesirable situation where _txCounter state variable will be initialized twice.

Recommendation:

If the above-mentioned scenario is not intended, the **_handleLottery** function must be redesigned to resolve the issue.

A.2 insertUser function is made PUBLIC and Accessible to users

Line no - 586-597

Explanation

The FortuneCookie V2 contract has a very crucial function, i.e., **insertUser** that initializes the **userData** struct and stores new users on the contract with the imperative details about the user like ***address, winning counts, index etc.***

```
586     function insertUser(address userAddress, uint winnings) public returns(uint256 index) {
587         if (_isExcludedFromLottery[userAddress]) {
588             return index;
589         }
590 
591         userByAddress[userAddress] = userData(userAddress, winnings, winnings, _countUsers, true);
592         userByIndex[_countUsers] = userData(userAddress, winnings, winnings, _countUsers, true);
593         index = _countUsers;
594         _countUsers += 1;
595 
596         return index;
597     }
```

However, the function has not been assigned any **onlyOwner** modifier and has been marked as **public,** which makes it accessible to all users. It will lead to an scenario where anyone can call this function with any arguments they want.

Recommendation:

If the above-mentioned scenario is not intended, it is recommended to make the **insertUser function** only accessible to the Owner of the contract.

A.3 The function design of *addWinner* is inadequate

Line no - 621-629

Explanation:

In the FortuneCookie V2 contract, the **addWinner** function performs a crucial task of updating the last winner's value as well as the address.

```
621     function addWinner(address userAddress, uint256 _lastWon) public returns (bool result) {
622         result = false;
623 
624         lastWinner_value = _lastWon;
625         lastWinner_address = userAddress;
626         result = true;
627 
628         return result;
629     }
```

However, despite the fact the fact that these are imperative state variables of the contract, the **addWinner** function is not designed effectively.

Mentioned below are the reason behind the poor design of **addWinner function:**

- The function has not been assigned an **onlyOwner** modifier which makes it accessible to every user.
- The function doesn't involve any **input validations** on the arguments passed to it.
- It involves a redundant variable update at Line 622 as every boolean variable is, by default, **false.**

Moreover, the **addWinner** function is being called once inside the **_handleLottery function.** Therefore, if the ***addWinner*** function is only supposed to be called from within the contract, the visibility keyword should be changed from **PUBLIC** to **INTERNAL.**

Recommendation:
The current function design of **addWinner** is not very effective. It is recommended to go through the above-mentioned issues and update the function accordingly, unless intended.

Medium Severity Issues

A.4 Loops are extremely costly

Line no -716, 797, 1251

Description:

The **FortuneCookie V2** 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:

- ***includeInReward***
- ***_getCurrentSupply***
- ***airDrop***

```
1252        for (uint256 index = 0; index < _recipients.length; index++) {
1253            if (!airdropReceived[_recipients[index]]) {
1254                airdropReceived[_recipients[index]] = true;
1255                transfer(_recipients[index], _amounts[index]);
1256                airdropped = airdropped.add(_amounts[index]);
1257            }
1258        }
```

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,

```
local_variable = _recipients.length
 for (uint256 index = 0; index < local_variable; index++) {
        if (!airdropReceived[_recipients[index]]) {
           airdropReceived[_recipients[index]] = true;
           transfer(_recipients[index], _amounts[index]);
           airdropped = airdropped.add(_amounts[index]);
        }
   }
```

A.5 No Input Validations performed on *insertUser* function

Line no - 586

Explanation:

The **insertUser** function initializes the **userData** struct with crucial information like address, index as well as win counts.

However, no input validation is performed on any of the arguments passed to this function. This will lead to a situation where ***zero addresses*** can be passed as user's address as well as any uint value can be passed for the ***totalWon or lastWon*** elements of the userStruct of the user being added in the contract.

Recommendation:

Including input validation checks ensures that no invalid arguments are passed while calling the function.

A.6 The *_handleLottery* function includes redundant IF Statement condition

Line no - 1198

Explanation:

The following IF statement in the ***handleLottery function*** includes a redundant conditional check with the **_balanceWinner** local variable.

```
1197
1198            if (_balanceWinner >= 0 && _balanceWinner >= _minBalance) {
1199
```

It ensures that the **balanceWinner** variable must be greater than zero **and** greater the minBalance.
However, the "**_balanceWinner >= 0"** validation is not necessary as it will automatically be ensured if the **balanceWinner is greater than the _minBalance** local variable.

Recommendation:
If the above mentioned scenario is not intended, it recommended to modify the IF statements accordingly.

Low Severity Issues

A.7 Comparison to boolean Constant

Line no: 580

Description:
Boolean constants can directly be used in conditional statements or require statements.
Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** statements.

```
577     function isUser(address userAddress) private view returns(bool isIndeed)
578     {
579         isIndeed = false;
580         if(userByAddress[userAddress].tokenOwner == true) {
581             isIndeed = true;
582         }
583         return isIndeed;
584     }
```

Recommendation:
The equality to boolean constants must be removed from the above-mentioned line.

A.8 Absence of Zero Address Validation

Line no-

Description:
The **FortuneCookie V2** contract includes quite a few functions that updates some of the imperative addresses in the contract like *lastWinner_address,uniswapV2Pair* ***etc***.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented on the following functions while updating the address state variables of the contract:

- ***addWinner***
- ***setUniswapPair***

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

A.9 External Visibility should be preferred

Explanation:
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:

- ***insertUser***
- ***getTotalWon***
- ***getLastWon***
- ***getCirculatingSupply***
- ***addWinner***
- ***getLastWinner***
- ***isExcludedFromReward***
- ***totalFees***
- ***deliver***
- ***reflectionFromToken***
- ***excludeFromReward***
- ***excludeFromFee***
- ***includeInFee***
- ***setSwapAndLiquifyEnabled***
- ***isExcludedFromFee***

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

A.10 Contract includes Hardcoded Addresses

Line no - 451-457
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.
However, the contract does include some hardcoded addresses in the above-mentioned lines.

```
451     address private constant _marketingWallet = 0xa1B01377fB1A7808f0e1211adA3867eBe211B142;
452
453     // An address used to transiently store the pot.
454     // We can store the pot in memory, but an address allows us
455     // to use the existing fee transfer abstractions as-is.
456     address private constant _potAddress = 0xf293cBd510b0875611cBc51225cE9A0790Ce168B;
457     address public immutable _burnAddress = 0x000000000000000000000000000000000000dEaD;
```

Recommendation:
of including hardcoded addresses in the contract, it would be an effective approach to initialize those addresses within the constructors at the time of deployment.

Informational

A.11 Coding Style Issues in the Contract

Explanation:
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.

```
Constant FortuneCookieV2._potAddress (contracts/FourtuneCookieV2.sol#456) is not in UPPER_CASE_WITH_UNDERSCORES
Variable FortuneCookieV2._burnAddress (contracts/FourtuneCookieV2.sol#457) is not in mixedCase
Variable FortuneCookieV2._taxFee (contracts/FourtuneCookieV2.sol#468) is not in mixedCase
Variable FortuneCookieV2._liquidityFee (contracts/FourtuneCookieV2.sol#469) is not in mixedCase
Variable FortuneCookieV2._marketingFee (contracts/FourtuneCookieV2.sol#470) is not in mixedCase
Variable FortuneCookieV2._burnFee (contracts/FourtuneCookieV2.sol#471) is not in mixedCase
Variable FortuneCookieV2._potFee (contracts/FourtuneCookieV2.sol#472) is not in mixedCase
Variable FortuneCookieV2._maxTxAmount (contracts/FourtuneCookieV2.sol#480) is not in mixedCase
Variable FortuneCookieV2.lastWinner_value (contracts/FourtuneCookieV2.sol#485) is not in mixedCase
Variable FortuneCookieV2.lastWinner_address (contracts/FourtuneCookieV2.sol#486) is not in mixedCase
```

During the automated testing, it was found that the **FortuneCookieV2** 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.

A.12 NatSpec Annotations must be included

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

Recommendation:
Cover by NatSpec all Contract methods.

Automated Test Results

```
totalFees() should be declared external:
        - FortuneCookieV2.totalFees() (contracts/FourtuneCookieV2.sol#669-671)
deliver(uint256) should be declared external:
        - FortuneCookieV2.deliver(uint256) (contracts/FourtuneCookieV2.sol#673-682)
reflectionFromToken(uint256,bool) should be declared external:
        - FortuneCookieV2.reflectionFromToken(uint256,bool) (contracts/FourtuneCookieV2.sol#684-694)
excludeFromReward(address) should be declared external:
        - FortuneCookieV2.excludeFromReward(address) (contracts/FourtuneCookieV2.sol#703-711)
excludeFromFee(address) should be declared external:
        - FortuneCookieV2.excludeFromFee(address) (contracts/FourtuneCookieV2.sol#727-729)
includeInFee(address) should be declared external:
        - FortuneCookieV2.includeInFee(address) (contracts/FourtuneCookieV2.sol#731-733)
setSwapAndLiquifyEnabled(bool) should be declared external:
        - FortuneCookieV2.setSwapAndLiquifyEnabled(bool) (contracts/FourtuneCookieV2.sol#764-767)
isExcludedFromFee(address) should be declared external:
        - FortuneCookieV2.isExcludedFromFee(address) (contracts/FourtuneCookieV2.sol#844-846)
```

```
FortuneCookieV2.allowance(address,address).owner (contracts/FourtuneCookieV2.sol#640) shadows:
        - Ownable.owner() (contracts/FourtuneCookieV2.sol#187-189) (function)
FortuneCookieV2._approve(address,address,uint256).owner (contracts/FourtuneCookieV2.sol#848) shadows:
        - Ownable.owner() (contracts/FourtuneCookieV2.sol#187-189) (function)
```

```
Ownable._previousOwner (contracts/FourtuneCookieV2.sol#176) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
Ownable._lockTime (contracts/FourtuneCookieV2.sol#177) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
FortuneCookieV2.numTokensSellToAddToLiquidity (contracts/FourtuneCookieV2.sol#481) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
FortuneCookieV2._maxWalletToken (contracts/FourtuneCookieV2.sol#482) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
FortuneCookieV2.w_rt (contracts/FourtuneCookieV2.sol#491) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
FortuneCookieV2._txCounter (contracts/FourtuneCookieV2.sol#492) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
FortuneCookieV2.transactionsSinceLastLottery (contracts/FourtuneCookieV2.sol#495) is never used in FortuneCookieV2 (contracts/FourtuneCookieV2.sol#428-1263)
```

```
Variable FortuneCookieV2._burnAddress (contracts/FourtuneCookieV2.sol#457) is not in mixedCase
Variable FortuneCookieV2._taxFee (contracts/FourtuneCookieV2.sol#468) is not in mixedCase
Variable FortuneCookieV2._liquidityFee (contracts/FourtuneCookieV2.sol#469) is not in mixedCase
Variable FortuneCookieV2._marketingFee (contracts/FourtuneCookieV2.sol#470) is not in mixedCase
Variable FortuneCookieV2._burnFee (contracts/FourtuneCookieV2.sol#471) is not in mixedCase
Variable FortuneCookieV2._potFee (contracts/FourtuneCookieV2.sol#472) is not in mixedCase
Variable FortuneCookieV2._maxTxAmount (contracts/FourtuneCookieV2.sol#480) is not in mixedCase
Variable FortuneCookieV2.lastWinner_value (contracts/FourtuneCookieV2.sol#485) is not in mixedCase
Variable FortuneCookieV2.lastWinner_address (contracts/FourtuneCookieV2.sol#486) is not in mixedCase
```