DreamLoops Smart Contract Final Audit Report

Project Synopsis

Project Name	**DreamLoops**
Platform	Ethereum, Solidity
Github Repo	Not Provided
Deployed Contract	Not Deployed
Total Duration	5 Days
Timeline of Audit	**29th April 2021 to 4th April 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	DreamPools.sol
Language	Solidity
Commit Hash	**Null**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	0	0
Medium Severity	0	3
Low Severity	1	3
Information	0	1
Total Found	1	7

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: DreamPools

High Severity Issues

None Found

Medium Severity Issues

A.1 Loops are extremely costly

Status: CLOSED

Line no - 61, 66

Description:

The **DreamLoops** 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.

```
61          for (uint256 i = 0; i < admins.length; i++) {
62              _setupRole(DEFAULT_ADMIN_ROLE, admins[i]);
63          }
64
```

As a result, these state variables consumes a lot more extra gas for every iteration of the for loop.

The following function includes such loops at the mentioned lines:

- ***constructor at Line 61,66***

Recommendation:

Its 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 = admins.length
for (uint256 i = 0; i < local_variable; i++) {
        _setupRole(DEFAULT_ADMIN_ROLE, admins[i]);
    }
```

A.2 State Variables Updated After External Call. Violation of Check_Effects_Interaction Pattern

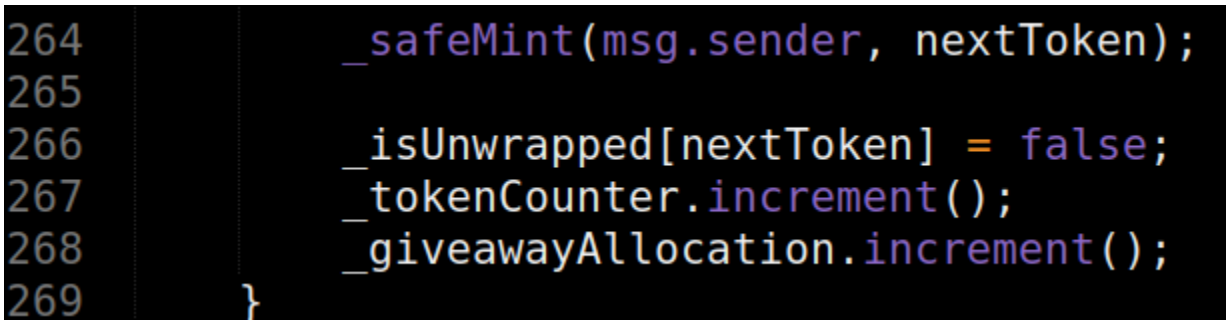
Status: CLOSED

Line - 264-269, 308-311, 322-325

Explanation:

As per the Check_Effects_Interaction Pattern in Solidity, external calls should be made at the very end of the function. Event emission as well as any state variable modification must be done before the external call is made.

However, during the manual review of the DreamLoops.sol contract, it was found that some of the functions in the contract violate this ***Check-Effects-Interaction*** pattern at the above-mentioned lines.



```
264             _safeMint(msg.sender, nextToken);
265
266             _isUnwrapped[nextToken] = false;
267             _tokenCounter.increment();
268             _giveawayAllocation.increment();
269         }
```

These functions update state variables in the contract after making an external call. Although the external call is made to the token contract itself, the secure development practices must not be avoided. The following are the functions that implement such function design:

- ***redeemGiveaway***
- ***mint***
- ***companyMint***

Recommendation:

Modification of any State Variables must be performed before making an external call. **Check Effects Interaction Pattern** must be followed while implementing external calls in a function.

A.3 Redundant Require Statements in Constructor of the Contract

Status: CLOSED

Line no : 51-55

Description:

a. **Constructor:** The constructor of the DreamLoops contract includes a require statement at the very top to ensure that the allocations are done as expected.

```
51      require(
52          MAX_TOKENS - MAX_PUBLIC - MAX_GIVEAWAYS - MAX_COMPANY_ALLOCATION ==
53              0,
54          "Does not equal"
55      );
```

However, since the allocations for Public, Giveaways etc are being hardcoded in the contract state variable itself(*from* *Line 39-42*), this **require statement** is not really significant.

```
39      uint256 public constant MAX_TOKENS = 100;
40      uint256 private constant MAX_PUBLIC = 85;
41      uint256 private constant MAX_GIVEAWAYS = 5;
42      uint256 private constant MAX_COMPANY_ALLOCATION = 10;
43
```

Such **require** statements checks are effective and necessary when allocation amounts are passsed as argument to the constructor.

b. addGiveawayUsers: The **addGiveawayUsers** function also includes a requirements to ensures that empty arrays are not being passed in the function.

```
237     function addGiveawayUsers(address[] memory giveawayUsers) public onlyAdmin {
238         //get array length
239         uint256 len = giveawayUsers.length;
240         require(len > 0, "Need to pass at least one address");
241
```

However, since the ***onlyAdmin*** modifier has already been attached to the function making it only accessible to the admins instead of all users, this require statement is not imperative as the admins of the contract should never pass empty arrays.

It must be noted that adding such unnecessary **require statements** will increase the gas consumption in during contract deployment which is not very adequate.

Recommendation:
Redundant **require statements** should be removed from the smart contract.

Low Severity Issues

A.4 External Visibility should be preferred

Status: CLOSED

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:

- ***grantRoleOwner***
- ***setBaseURI***
- ***setSale***
- ***isSaleStarted***
- ***calculatePriceForToken***
- ***addGiveawayUsers***
- ***redeemGiveaway***
- ***mint***
- ***companyMint***
- ***withdrawAll***

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

A.5 Comparison to boolean Constant

Status: Partially CLOSED

Line no: 294
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.

```
292     function mint(uint256 quantity) public payable whenNotPaused {
293         require(quantity > 0, "You must purchase at least one token");
294         require(saleState == true, "Sale has not started");
295
```

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

A.6 Absence of Error messages in Require Statements

Status: CLOSED

Line no - 77, 332
Description:
The ***DreamLoops*** contract includes a few functions(*at the above-mentioned lines*) that doesn't contain any error message in the **require** statement.

```
332             require(payable(msg.sender).send(address(this).balance));
333     }
334 }
```

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

A.7 Redundant Initialization of State Variable

Status: CLOSED

Line no - 57
Description:
The State Variable **_paused** is being initialized to **FALSE** in the constructor.

```
56          _tokenCounter.increment();
57          _paused = false;
58          _baseTokenURI = _newBaseURI;
59
```

However, boolean state variables are **FALSE** by default and does not require explicit initialization to false.

Recommendation:
Unnecessary Initialization of State variables should be avoided in the contract.

Informational

A.8 Coding Style Issues in the Contract

Status: CLOSED

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.

```
Parameter Bitlectro.setBaseURI(string)._newBaseURI (flatDreams.sol#1741) is not in mixedCase
Parameter Bitlectro.bondingCurve(uint256)._id (flatDreams.sol#1785) is not in mixedCase
Parameter Bitlectro.calculatePriceForToken(uint256)._id (flatDreams.sol#1808) is not in mixedCase
```

During the automated testing, it was found that the DreamLoops contract had 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.