

Bitlectro Labs

DreamLoops

**Smart Contract Audit
Final Report**



May 08, 2021

Introduction	3
About Bitlectro Labs	3
About ImmuneBytes	3
Documentation Details	3
Audit Process & Methodology	4
Audit Details	4
Audit Goals	5
Security Level References	5
High severity issues	6
Medium severity issues	6
Low severity issues	9
Recommendations	11
Automated Test Result	12
Concluding Remarks	13
Disclaimer	13

This 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.

Introduction

1. About Bitlectro Labs

Bitlectro Labs Dreamloops are a programmatically generated series of audio-visual non-fungible tokens (NFTs). Minted on the Ethereum blockchain, and hosted via IPFS (Interplanetary File System), the first line of Dreamloops NFTs will feature 16-bit retro style musical loops. Each NFT has a one of a kind album art. 1 in 5 Dreamloops (20%) will be redeemable for physical media. 10,000 Dreamloops will be minted in the first release. As the first release from Bitlectro Labs, Dreamloops will be a set of chiptune style musical loops with unique melodies, drum beats, and basslines. Each tune will have a corresponding cover or album art, designed in a retro 16-bit fashion but with unique visual elements - think retro-futurism. The elements and features of both the music and cover art will have a fixed scarcity. Some covers will be fully animated and have features unique to only a few (or in some instances one) Dreamloops in the edition, whereas others will be common and easier to obtain. Musical compositions will be shared across the collection, with some only appearing with “rare” or “uncommon” NFTs and others being more obtainable with “common” releases.

Visit <http://bitlectrolabs.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 Bitlectro Labs team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.google.com/document/d/12TkzNNCo_yLhTZc3KMRF33qWg3mQq5dMXu7Xz-vPJn4/edit

This 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.

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: Bitlectro Labs
- Contract Name: Dreamloops.sol
- Languages: Solidity(Smart contract)
- Github commit hash for audit: [a58898c71e868d32138b715fc02dd40ab7251459](#)
- GitHub link: https://github.com/tigerthelion/bitlectro_contract_v1
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

This 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.

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	<u>High</u>	<u>Medium</u>	<u>Low</u>
Open	-	-	1
Closed	-	3	3

This 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.

High severity issues

No issues found.

Medium severity issues

1. Loops are extremely costly

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:

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 = admins.length
```

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

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/DreamLoops/contracts/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

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

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

This 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.

Description:

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.

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/DreamLoops/dreamloops/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

3. Redundant Require Statements in Constructor of the Contract

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.

This 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.

```

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 passed 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 during contract deployment which is not very adequate.

Recommendation:

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

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/DreamLoops/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

Low severity issues

1. 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:

- **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.

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/0xPunks/0xPunks/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

2. Comparison to boolean Constant

Line no: 289, 298

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.

```
288     );  
289     require(isUnwrapped(tokenId) == false, "Token is already unwrapped");  
290     _isUnwrapped[tokenId] = true;  
291  
292
```

This 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.

Recommendation:

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

Status: Partially CLOSED

3. Absence of Error messages in Require Statements

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

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/DreamLoops/contract/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

4. Redundant Initialization of State Variable

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.

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/dreamloops/flatdreams/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

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.

```
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.

Amended (May 8th 2021): Issue was fixed by DreamLoops team and is no longer present in the code. GitHub Commit number:

[f8ada1c85f216270fa0fc481dafd6348cc61fe20](https://github.com/dreamloops/flatdreams/commit/f8ada1c85f216270fa0fc481dafd6348cc61fe20)

Automated Test Result

```

Reentrancy in Bitlectro.companyMint(address) (flatDreams.sol#1918-1926):
  External calls:
    - _safeMint(to,nextToken) (flatDreams.sol#1922)
      - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
  State variables written after the call(s):
    - _isUnwrapped[nextToken] = false (flatDreams.sol#1923)
Reentrancy in Bitlectro.mint(uint256) (flatDreams.sol#1896-1913):
  External calls:
    - _safeMint(msg.sender,nextToken) (flatDreams.sol#1908)
      - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
  State variables written after the call(s):
    - _isUnwrapped[nextToken] = false (flatDreams.sol#1909)
Reentrancy in Bitlectro.redeemGiveaway() (flatDreams.sol#1852-1869):
  External calls:
    - _safeMint(msg.sender,nextToken) (flatDreams.sol#1864)
      - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
  State variables written after the call(s):
    - _isUnwrapped[nextToken] = false (flatDreams.sol#1866)

grantRoleOwner(bytes32,address) should be declared external:
  - Bitlectro.grantRoleOwner(bytes32,address) (flatDreams.sol#1722-1728)
setBaseURI(string) should be declared external:
  - Bitlectro.setBaseURI(string) (flatDreams.sol#1741-1743)
setSale() should be declared external:
  - Bitlectro.setSale() (flatDreams.sol#1766-1768)
isSaleStarted() should be declared external:
  - Bitlectro.isSaleStarted() (flatDreams.sol#1775-1777)
calculatePriceForToken(uint256) should be declared external:
  - Bitlectro.calculatePriceForToken(uint256) (flatDreams.sol#1808-1810)
addGiveawayUsers(address[]) should be declared external:
  - Bitlectro.addGiveawayUsers(address[]) (flatDreams.sol#1837-1846)
redeemGiveaway() should be declared external:
  - Bitlectro.redeemGiveaway() (flatDreams.sol#1852-1869)
unwrapToken(uint256) should be declared external:
  - Bitlectro.unwrapToken(uint256) (flatDreams.sol#1883-1891)
mint(uint256) should be declared external:
  - Bitlectro.mint(uint256) (flatDreams.sol#1896-1913)
companyMint(address) should be declared external:
  - Bitlectro.companyMint(address) (flatDreams.sol#1918-1926)
withdrawAll() should be declared external:
  - Bitlectro.withdrawAll() (flatDreams.sol#1931-1933)

Bitlectro.unwrapToken(uint256) (flatDreams.sol#1883-1891) compares to a boolean constant:
  - require(bool,string)(isUnwrapped(tokenId) == false,Token is already unwrapped) (fl
Bitlectro.mint(uint256) (flatDreams.sol#1896-1913) compares to a boolean constant:
  - require(bool,string)(saleState == true,Sale has not started) (flatDreams.sol#1898)
Reference: https://github.com/crvtic/slither/wiki/Detector-Documentation#boolean-equality
Bitlectro.bondingCurve(uint256) (flatDreams.sol#1785-1791) uses literals with too many digits:
  - 2000000000000000000 (flatDreams.sol#1787)
Bitlectro.bondingCurve(uint256) (flatDreams.sol#1785-1791) uses literals with too many digits:
  - 1000000000000000000 (flatDreams.sol#1789)

```

This 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.

Concluding Remarks

While conducting the audits of Bitlectro Labs smart contract - DreamLoops.sol, 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 Bitlectro Labs developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Note: DreamLoops team has fixed the Medium and Low issues based on the auditor's recommendation.

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 Bitlectro Labs platform or its product neither 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 Pvt Ltd.