UniqDrop Smart Contract

Initial Audit Report

Project Synopsis

Project Name	**UniqDrop**
Platform	Ethereum, Solidity
Github Repo	https://github.com/uniqly-io/uniq-drop
Total Duration	4 Days
Timeline of Audit	**8th May 2021 to 11th May 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	UniqDrop
Language	Solidity
Commit Hash	**6f6a5b2fa266a2b1d95ab01578f9a97ea68b081a**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	3	0
Medium Severity	1	0
Low Severity	3	0
Information	2	0
Total Found	9	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: UniqDrop

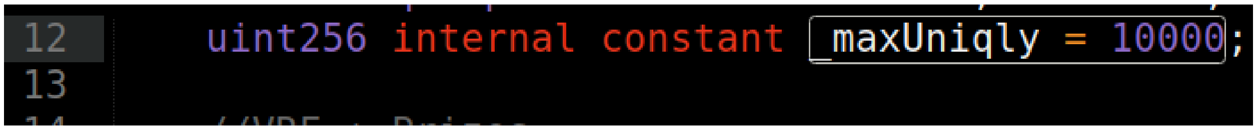
High Severity Issues

A.1 More UniqlyNFT Tokens can be minted than the MAXIMUM ALLOWED Tokens(_maxUniqly)

Line no - 293 to 304

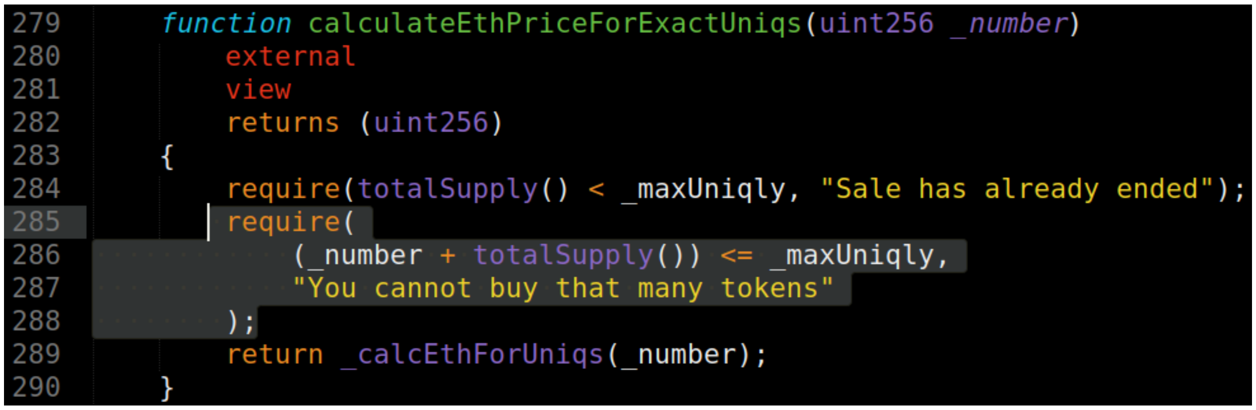
Description:

As per the current architecture of the contract, the *maximum amount of tokens* that should be minted is **10000**.



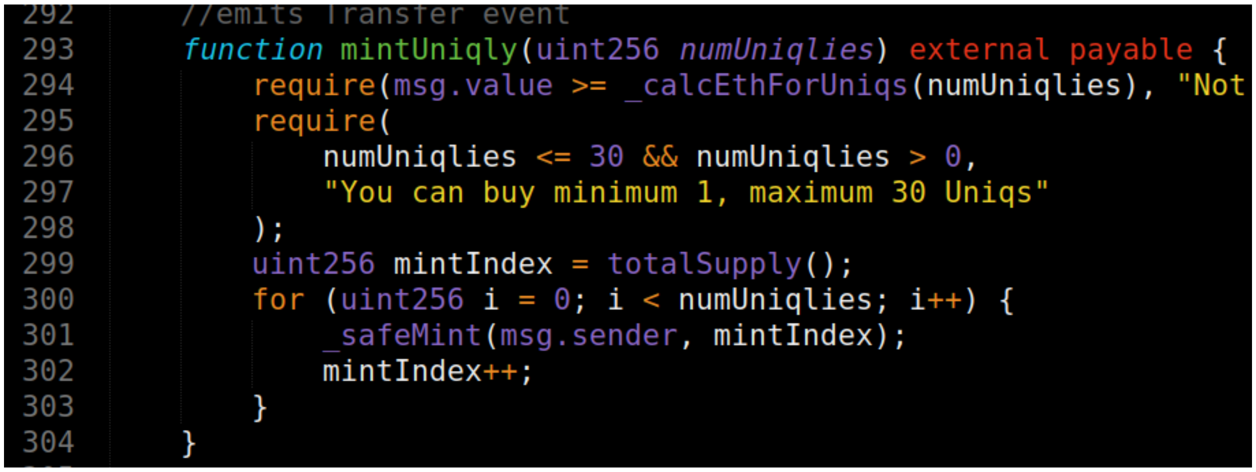
```
    uint256 internal constant _maxUniqly = 10000;
```

In order to ensure that no tokens above this amount is minted, adequate validations have been included in the calculateEthPriceForExactUniqs function(Line 284-288).



```
279     function calculateEthPriceForExactUniqs(uint256 _number)
280         external
281         view
282         returns (uint256)
283     {
284         require(totalSupply() < _maxUniqly, "Sale has already ended");
285         require(
286             (_number + totalSupply()) <= _maxUniqly,
287             "You cannot buy that many tokens"
288         );
289         return _calcEthForUniqs(_number);
290     }
```

However, no such validations have been implemented in the mintUniqly function to ensure that the **totalSupply** does not exceed the ***_maxUniqly*** amount.



```
292        //emits Transfer event
293        function mintUniqly(uint256 numUniqlies) external payable {
294            require(msg.value >= _calcEthForUniqs(numUniqlies), "Not
295            require(
296                numUniqlies <= 30 && numUniqlies > 0,
297                "You can buy minimum 1, maximum 30 Uniqs"
298            );
299            uint256 mintIndex = totalSupply();
300            for (uint256 i = 0; i < numUniqlies; i++) {
301                _safeMint(msg.sender, mintIndex);
302                mintIndex++;
303            }
304        }
```

Although the mintUniqly function includes a require statement to check that the function argument ***numUniqlies*** should not be between 0 to 30(*Line* 295-298), it does not take into consideration the following scenario:

- If the total Supply is already at 9990 and a user wishes to mint 30 new tokens, then the total supply will exceed the _maxUniqly amount.

Is this INTENDED?

Recommendation:

If the above mentioned scenario is not intended, then a **require** statement must be implemented to ensure total supply always remains within the predefined limit.

An effective way of doing so is to use ***calculateEthPriceForExactUniqs*** function instead of ***_calcEthForUniqs*** function in the **require** statement of ***mintUniqly*** function at ***Line 294***. This is because the calculateEthPriceForExactUniqs already implements all the significant **require** statements in this case and ensures that the maximum **totalSupply** of tokens is always as expected.

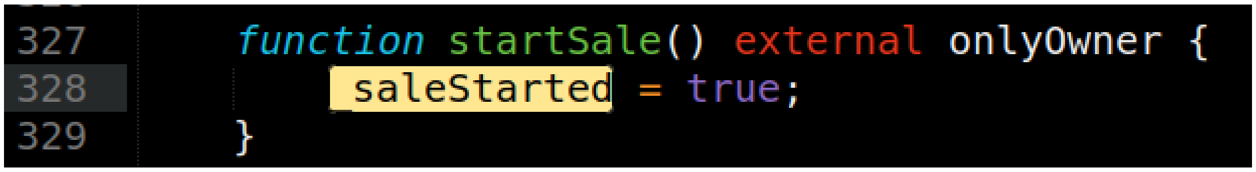
Moreover, if there is no such restrictions on the **totalSupply** of the token and tokens can be minted upto any maximum amount, then the contract should be updated accordingly.

A.2 New Uniqly Tokens can be Minted even before the SALE has Started

Line no - 293 to 304

Description:

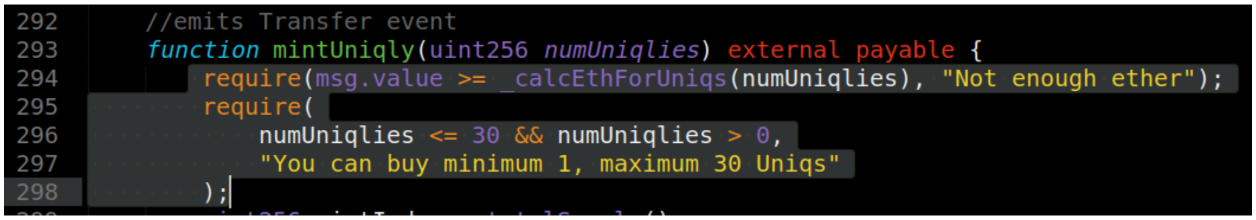
The current contract design allows the owner to start the sale(*Line* 327-239) but doesn't use this information effectively in the contract.



```
327     function startSale() external onlyOwner {
328         saleStarted = true;
329     }
```

For instance, the ***mintUniqly*** function can be accessed and users can start investing as well as minting new tokens even before the owner triggers the **startSale** function and actually initiates the sale process.

This is mainly because the **mintUniqly function** doesn't include any **require statements** to validate whether or not the **saleStarted** boolean is **TRUE.**



```
292     //emits Transfer event
293     function mintUniqly(uint256 numUniqlies) external payable {
294         require(msg.value >= _calcEthForUniqs(numUniqlies), "Not enough ether");
295         require(
296             numUniqlies <= 30 && numUniqlies > 0,
297             "You can buy minimum 1, maximum 30 Uniqs"
298         );
```

Recommendation:

In order to avoid the above-mentioned scenario, a **require statement must be included in the mintUniqly** function at the very beginning, as follows:

require(_saleStarted, "ERROR MSG: Sale has not Started Yet")

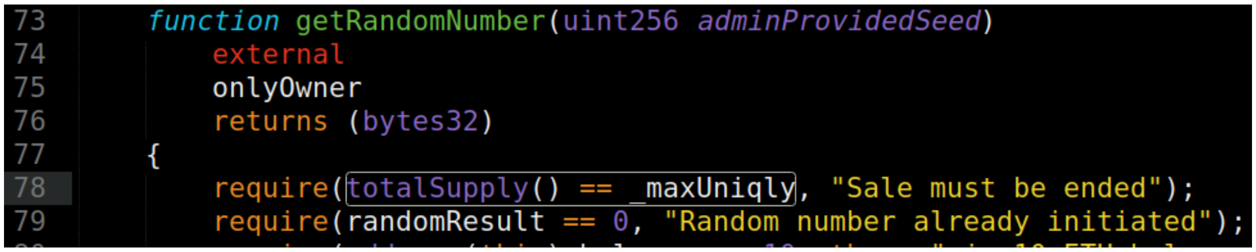
A.3 getRandomNumber function includes a Strict Equality Check

Line no - 78

Description:

The getRandomNumber function includes a strict equality check between totalSupply and _maxUniqly at the very start of the function body.

In order to execute this function, this require statement must be satisfied.



```
73      function getRandomNumber(uint256 adminProvidedSeed)
74          external
75          onlyOwner
76          returns (bytes32)
77      {
78          require(totalSupply() == _maxUniqly, "Sale must be ended");
79          require(randomResult == 0, "Random number already initiated");
```

However, since there is a strict equality check, the function becomes completely inaccessible even if the totalSupply() value is slightly above the _maxUniqly.

Is this Scenario INTENDED?

Recommendation:

If the above-mentioned scenario is not intended, the require statement can be modified as follows:

require(totalSupply() >= _maxUniqly, "Sale must be ended");

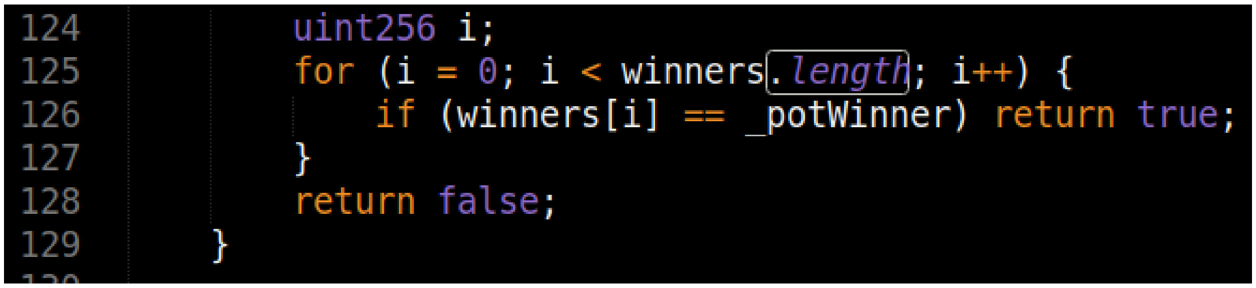
Medium Severity Issues

A.4 Loops are extremely costly

Line no - 61, 66

Description:

The **UniqDrop** contract has a **for loop** in the contract that includes state variables like .length of a non-memory array, in the condition of the for loops.



```
124            uint256 i;
125            for (i = 0; i < winners.length; i++) {
126                if (winners[i] == _potWinner) return true;
127            }
128            return false;
129        }
```

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 mentioned lines:

- ***_isAlreadyRecceivedPrize at Line 125***

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 = winners.length;
 for (i = 0; i <local_variable; i++) {
          if (winners[i] == _potWinner) return true;
      }
```

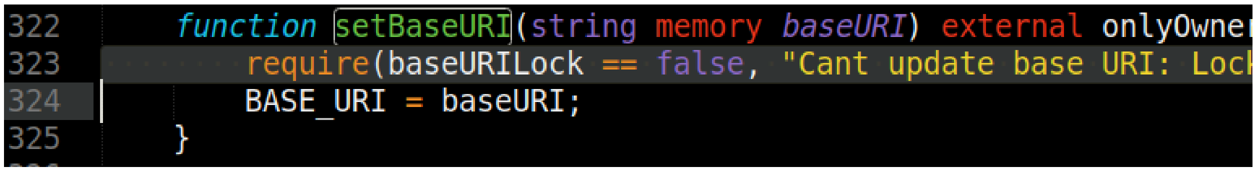
Low Severity Issues

A.5 Comparison to boolean Constant

Line no: 89, 135, 136-138, 323

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.



```
322     function setBaseURI(string memory baseURI) external onlyOwne
323         require(baseURILock == false, "Cant update base URI: Loc
324         BASE_URI = baseURI;
325     }
```

Recommendation:

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

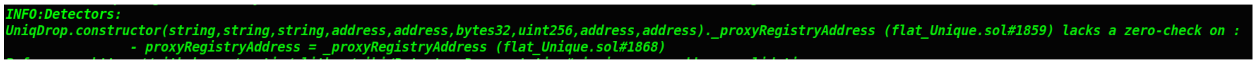
A.6 constructor does not include Zero Address Validation

Line no: 250-253

Explanation:

The **constructor** initializes one of the most imperative state variables, i.e., ***proxyRegistryAddress*** in the UniqDrop contract.

However, during the automated testing of the contract, it was found that the constructor doesn't implement any Zero Address Validation Check to ensure that no zero address is passed while initializing this state variable.



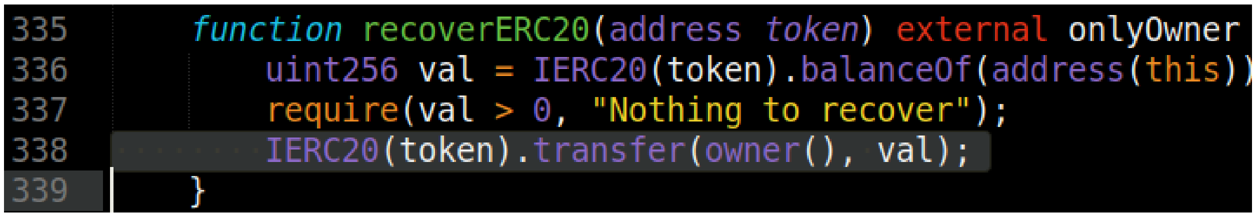
```
INFO:Detectors:
UniqDrop.constructor(string,string,string,address,address,bytes32,uint256,address,address)._proxyRegistryAddress (flat_Unique.sol#1859) lacks a zero-check on :
                - proxyRegistryAddress = _proxyRegistryAddress (flat_Unique.sol#1868)
```

A.7 Return Value of an External Call is never used Effectively

Line no - 338

Explanation:

The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.
These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.
However, the UniqDrop contract never uses these return values throughout the contract.



```
    function recoverERC20(address token) external onlyOwner
        uint256 val = IERC20(token).balanceOf(address(this))
        require(val > 0, "Nothing to recover");
        IERC20(token).transfer(owner(), val);
    }
```

Recommendation:

Effective use of all the return values from external calls must be ensured within the contract.

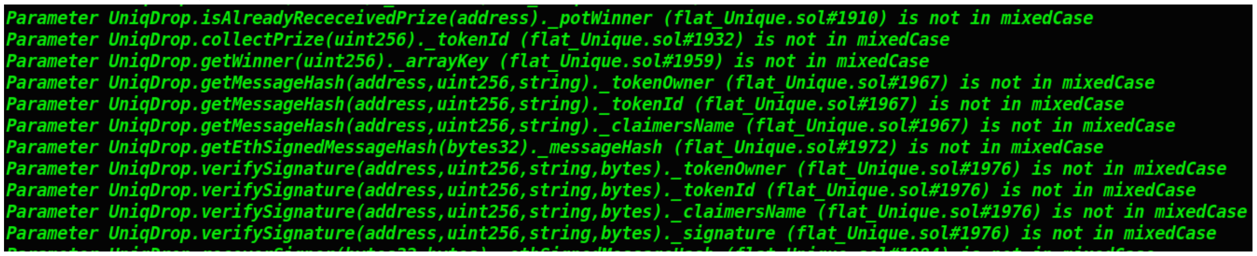
Informational

A. 8Coding 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.

During the automated testing, it was found that the UniqDrop contract had quite a few code style issues.



```
Parameter UniqDrop.isAlreadyRececeivedPrize(address)._potWinner (flat_Unique.sol#1910) is not in mixedCase
Parameter UniqDrop.collectPrize(uint256)._tokenId (flat_Unique.sol#1932) is not in mixedCase
Parameter UniqDrop.getWinner(uint256)._arrayKey (flat_Unique.sol#1959) is not in mixedCase
Parameter UniqDrop.getMessageHash(address,uint256,string)._tokenOwner (flat_Unique.sol#1967) is not in mixedCase
Parameter UniqDrop.getMessageHash(address,uint256,string)._tokenId (flat_Unique.sol#1967) is not in mixedCase
Parameter UniqDrop.getMessageHash(address,uint256,string)._claimersName (flat_Unique.sol#1967) is not in mixedCase
Parameter UniqDrop.getEthSignedMessageHash(bytes32)._messageHash (flat_Unique.sol#1972) is not in mixedCase
Parameter UniqDrop.verifySignature(address,uint256,string,bytes)._tokenOwner (flat_Unique.sol#1976) is not in mixedCase
Parameter UniqDrop.verifySignature(address,uint256,string,bytes)._tokenId (flat_Unique.sol#1976) is not in mixedCase
Parameter UniqDrop.verifySignature(address,uint256,string,bytes)._claimersName (flat_Unique.sol#1976) is not in mixedCase
Parameter UniqDrop.verifySignature(address,uint256,string,bytes)._signature (flat_Unique.sol#1976) is not in mixedCase
```

Recommendation:

Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

A.9 NatSpec Annotations must be included

Description:

The smart contracts do not include the NatSpec annotations adequately.

Recommendation:

Cover by NatSpec all Contract methods.