Horizen - Airdrop

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 3rd, 2022 - February 7th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/03/2022	Roberto Reigada
0.2	Document Updates	02/07/2022	Roberto Reigada
0.3	Draft Review	02/07/2022	Gabi Urrutia
1.0	Remediation Plan	02/15/2022	Roberto Reigada
1.1	Remediation Plan Review	02/16/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Horizen Labs engaged Halborn to conduct a security audit on their Airdrop smart contract beginning on February 3rd, 2022 and ending on February 7th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository HorizenLabs/grapes-erc20.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were addressed by Horizen Labs team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.

2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL

9 - 8 - HIGH

7 - 6 - MEDIUM

5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contract:

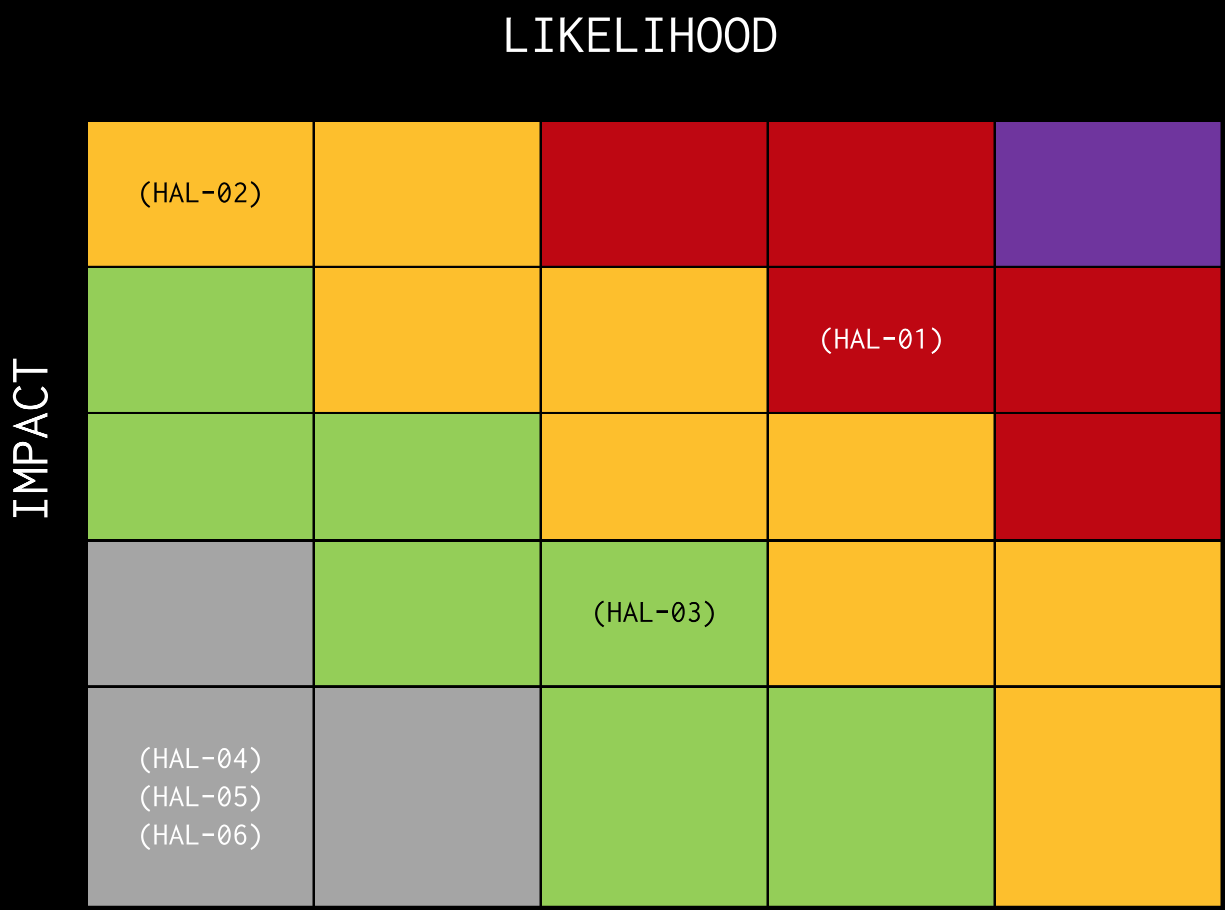
- AirdropGrapesToken.sol

Commit ID: 941af30005fdf0079207ad025c41f0571d261cf9

Fixed Commit ID: 498b5453953b33e142947f4a9365ccc88449085d

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	1	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - GAMMA TOKENS ARE SET AS CLAIMED EVEN WHEN THEY PROVIDE NO BONUS REWARDS	High	SOLVED - 02/15/2022
HAL02 - DOS WITH BLOCK GAS LIMIT	Medium	SOLVED - 02/15/2022
HAL03 - MISSING ZERO ADDRESS CHECKS	Low	SOLVED - 02/15/2022
HAL04 - UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	SOLVED - 02/15/2022
HAL05 - USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 02/15/2022
HAL06 - STATE VARIABLES MISSING IMMUTABLE MODIFIER	Informational	SOLVED - 02/15/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) GAMMA TOKENS ARE SET AS CLAIMED EVEN WHEN THEY PROVIDE NO BONUS REWARDS - HIGH

Description:

The gamma tokens owned will only increase the amount of tokens that can be claimed in the case that the gamma token can be paired with another alpha/beta token owned, as it is specified in the line 167 of the contract: `uint256 gammaToBeClaim = min(unclaimedAlphaBalance + unclaimedBetaBalance , unclaimedGamaBalance);`

This means that a user who holds 1 alpha NFT and 2 gamma NFTs will always get his grapes token amount increased once by 669.627, not twice, getting 6520.054 grapes tokens airdropped because of his alpha NFT and an extra 669.627 tokens because of one of the gamma NFTs he holds.

The problem with this scenario is that the second gamma NFT of the user would also be set as claimed in the smart contract, even though that NFT did not provide him with any reward.

Proof of Concept:

```
contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 100000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
contract_alpha.balanceOf(user1) -> 1
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 2
contract_AirdropGrapesToken.alphaClaimed(1) -> False
contract_AirdropGrapesToken.gammaClaimed(1) -> False
contract_AirdropGrapesToken.gammaClaimed(2) -> False
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0xf90229ba5a7afab75b84dfdcee35e3f502d7fde714f0ca689874b2dd2e54284d
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14135009   Gas used: 225278 (0.03%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 99992810319000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 7189681000000000000000
contract_AirdropGrapesToken.alphaClaimed(1) -> True
contract_AirdropGrapesToken.gammaClaimed(1) -> True
contract_AirdropGrapesToken.gammaClaimed(2) -> True
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to modify the claimTokens() function logic in order that it only marks as claimed those gamma NFTs that were actually providing a bonus in the airdrop.

Remediation Plan:

SOLVED: The Horizen Labs team addressed the issue in the claimTokens() function. Now, only those gamma NFTs that are actually providing a bonus in the airdrop are marked as claimed:

```
contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 100000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
contract_alpha.balanceOf(user1) -> 1
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 2
contract_AirdropGrapesToken.alphaClaimed(1) -> False
contract_AirdropGrapesToken.gammaClaimed(1) -> False
contract_AirdropGrapesToken.gammaClaimed(2) -> False
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x4b0f02f6179d42724b9bf4597a4faefa6b40ad35bc433d1b4263790e3f1967da
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14216666   Gas used: 185496 (0.02%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 99992810319000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 718968100000000000000
contract_AirdropGrapesToken.alphaClaimed(1) -> True
contract_AirdropGrapesToken.gammaClaimed(1) -> True
contract_AirdropGrapesToken.gammaClaimed(2) -> False
```

3.2 (HAL-02) DOS WITH BLOCK GAS LIMIT - MEDIUM

Description:

In the contract AirdropGrapesToken the function claimTokens(), as its name indicates, is used by the different NFT holders to claim their corresponding grapes tokens. This function iterates over the different tokenIds held by the caller, setting them as claimed so no user will be able to claim for the same tokenId twice.

In the case that the user calling this function is, for example, a whale that holds more than around 800 NFTs of any supported type in total, the gas costs of this function call would exceed the Ethereum mainnet block gas limit of 30 millions (block gas limit in Polygon is actually much lower than this). This means that the transaction would never get processed by a miner, making the user waste a lot of gas and not allowing the user to claim his grapes tokens:

```
contract_alpha.balanceOf(user1) -> 300
contract_beta.balanceOf(user1) -> 300
contract_gamma.balanceOf(user1) -> 300
contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 1000000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0xa2e1ded2eef3c7abd88dbc55e249d19e59f04b93bd56daeb60e946477734b7f7
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14135596   Gas used: 35713125

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 974175301000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 25824699000000000000000000
```

Code Location:

Listing 1: AirdropGrapesToken.sol (Lines 105,113,122)

```solidity
function claimTokens() external whenNotPaused {
    require(block.timestamp >= claimStartTime && block.timestamp <
        claimStartTime + claimDuration, "Claimable period is
        finished");
    require((beta.balanceOf(msg.sender) > 0 || alpha.balanceOf(msg
        .sender) > 0), "Nothing to claim");

```

```
    uint256 tokensToClaim;
    uint256 gammaToBeClaim;

    (tokensToClaim, gammaToBeClaim) = getClaimableTokenAmountAndGammaToClaim(msg.sender);

    for(uint256 i = 0; i < alpha.balanceOf(msg.sender); i++) {
        uint256 tokenId = alpha.tokenOfOwnerByIndex(msg.sender, i);
        if(!alphaClaimed[tokenId]) {
            alphaClaimed[tokenId] = true;
            emit AlphaClaimed(tokenId, msg.sender, block.timestamp);
        }
    }

    for(uint256 i = 0; i < beta.balanceOf(msg.sender); i++) {
        uint256 tokenId = beta.tokenOfOwnerByIndex(msg.sender, i);
        if(!betaClaimed[tokenId]) {
            betaClaimed[tokenId] = true;
            emit BetaClaimed(tokenId, msg.sender, block.timestamp);
        }
    }

    uint256 currentGammaClaimed;
    for(uint256 i = 0; i < gamma.balanceOf(msg.sender); i++) {
        uint256 tokenId = gamma.tokenOfOwnerByIndex(msg.sender, i);
        if(!gammaClaimed[tokenId] && currentGammaClaimed <= gammaToBeClaim) {
            gammaClaimed[tokenId] = true;
            emit GammaClaimed(tokenId, msg.sender, block.timestamp);
            currentGammaClaimed++;
        }
    }

    grapesToken.safeTransfer(msg.sender, tokensToClaim);

    totalClaimed += tokensToClaim;
    emit AirDrop(msg.sender, tokensToClaim, block.timestamp);
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

It is recommended to add a new function into the AirdropGrapesToken contract that allows the user to claim for a bundle of tokenIds and not all the held NFTs at once.

Remediation Plan:

SOLVED: No functionality was added to the smart contract to address this. Although, Halborn and Horizen Labs team checked that the current top holder of alpha, beta, and gamma tokens got a total of 310 NFTs. This should not suppose a problem as the gas costs of calling claimTokens() with 310 NFTs are way below of the current block gas limit of 30.000.000 for the Ethereum mainnet. Some tests were executed, getting the results below:

1. A user with a total of 300 NFTs will pay 10.984.638 gas to perform the claimTokens() call:

```
contract_alpha.balanceOf(user1) -> 300
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 0

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 1000000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x3bba7c0970cec4a4437961d297a079efef788405f932f154fe2a0b7ac4a852d4
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14216732   Gas used: 10984638 (1.37%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 980439838000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 19560162000000000000000000
```

2. A user with a total of 600 NFTs will pay 21.875.674 gas to perform the claimTokens() call:

FINDINGS & TECH DETAILS

```
contract_alpha.balanceOf(user1) -> 600
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 0

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 1000000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x61b4ca754df250f8ededd139819b5c30c861f4e4cdbd8dae629ebcbaa5f563b1
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 1
  AirdropGrapesToken.claimTokens confirmed   Block: 14217341   Gas used: 21875674 (2.73%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 960879676000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 39120324000000000000000000
```

3. A user with a total of 800 NFTs will pay 29.139.490 gas to perform the claimTokens() call:

```
contract_alpha.balanceOf(user1) -> 800
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 0

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 1000000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0xc192af8ecc158e8093624e6b164c4fc0e8f79a790588899ff63f7daa7790185c
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14217284   Gas used: 29139490 (3.64%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 947839568000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 52160432000000000000000000
```

As long as there are no holders with more than 800 NFTs in their wallet, there should be no issue in the claimTokens() call. Although, if that was the case, the user could also transfer some of his NFTs to another wallet and perform the claim from 2 different wallets as a mitigation.

3.3 (HAL-03) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The constructor of the AirdropGrapesToken contract is missing address validation. Every address should be validated and checked that is different from zero. This is also considered a best practice.

Code Location:

Listing 2: AirdropGrapesToken.sol (Lines 62-65,70-73)

```
61 constructor(
62     address _grapesTokenAddress,
63     address _alphaContractAddress,
64     address _betaContractAddress,
65     address _gammaContractAddress,
66     uint256 _ALPHA_DISTRIBUTION_AMOUNT,
67     uint256 _BETA_DISTRIBUTION_AMOUNT,
68     uint256 _GAMMA_DISTRIBUTION_AMOUNT
69 ) {
70     grapesToken = IERC20(_grapesTokenAddress);
71     alpha = ERC721Enumerable(_alphaContractAddress);
72     beta = ERC721Enumerable(_betaContractAddress);
73     gamma = ERC721Enumerable(_gammaContractAddress);
74     ALPHA_DISTRIBUTION_AMOUNT = _ALPHA_DISTRIBUTION_AMOUNT;
75     BETA_DISTRIBUTION_AMOUNT = _BETA_DISTRIBUTION_AMOUNT;
76     GAMMA_DISTRIBUTION_AMOUNT = _GAMMA_DISTRIBUTION_AMOUNT;
77
78     _pause();
79 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

SOLVED: The Horizen Labs team now validates that every address input is different from zero.

3.4 (HAL-04) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0 - INFORMATIONAL

Description:

`uint256` variables are already initialized to 0 by default. `uint256 i = 0` would reassign the 0 to `i` which wastes gas.

Code Location:

AirdropGrapesToken.sol

- Line 105: `for(uint256 i = 0; i < alpha.balanceOf(msg.sender); i++){`
- Line 113: `for(uint256 i = 0; i < beta.balanceOf(msg.sender); i++){`
- Line 122: `for(uint256 i = 0; i < gamma.balanceOf(msg.sender); i++){`
- Line 146: `for(uint256 i = 0; i < alpha.balanceOf(_account); i++){`
- Line 153: `for(uint256 i = 0; i < beta.balanceOf(_account); i++){`
- Line 160: `for(uint256 i = 0; i < gamma.balanceOf(_account); i++){`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize uint256 variables to 0 to save some gas. For example, use instead:
`for(uint256 i; i < alpha.balanceOf(msg.sender); i++){`.

Remediation Plan:

SOLVED: The initialization of uint256 variables to 0 was removed by Horizen Labs team to save some gas.

3.5 (HAL-05) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In all the loops, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`. This also affects variables incremented inside the loop code block.

Code Location:

AirdropGrapesToken.sol
- Line 105: `for(uint256 i = 0; i < alpha.balanceOf(msg.sender); i++){`
- Line 113: `for(uint256 i = 0; i < beta.balanceOf(msg.sender); i++){`
- Line 122: `for(uint256 i = 0; i < gamma.balanceOf(msg.sender); i++){`
- Line 146: `for(uint256 i = 0; i < alpha.balanceOf(_account); i++){`
- Line 149: `unclaimedAlphaBalance++;`
- Line 153: `for(uint256 i = 0; i < beta.balanceOf(_account); i++){`
- Line 156: `unclaimedBetaBalance++;`
- Line 160: `for(uint256 i = 0; i < gamma.balanceOf(_account); i++){`
- Line 163: `unclaimedGamaBalance++;`

Proof of Concept:

For example, based in the following test contract:

Listing 3: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
```

```
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```

```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop. This also applies to the variables declared inside the for loop, not just the iterator. On the other hand, this is not applicable outside of loops.

For example:

Listing 4: AirdropGrapesToken.sol (Lines 146,149)

```
for(uint256 i; i < alpha.balanceOf(_account); ++i) {
    uint256 tokenId = alpha.tokenOfOwnerByIndex(_account, i);
```

```
    if(!alphaClaimed[tokenId]) {
        ++unclaimedAlphaBalance;
    }
}
```

Remediation Plan:

SOLVED: The Horizen Labs team uses now ++i to increment the variables inside loops, saving some gas.

3.6 (HAL-06) STATE VARIABLES MISSING IMMUTABLE MODIFIER - INFORMATIONAL

Description:

The immutable keyword was added to Solidity in 0.6.5. State variables can be marked immutable which causes them to be read-only, but only assignable in the constructor. The following state variables are missing the immutable modifier:

AirdropGrapesToken.sol
- Line 12: IERC20 public grapesToken;;
- Line 14: ERC721Enumerable public alpha;
- Line 15: ERC721Enumerable public beta;
- Line 16: ERC721Enumerable public gamma;

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add the immutable modifier to the state variables mentioned to save some gas.

Remediation Plan:

SOLVED: The Horizen Labs team added the immutable modifier to the state variables suggested.

MANUAL TESTING

Halborn performed different manual tests in the AirdropGrapesToken contract to check for any logic flaws.

4.1 CLAIMED TOKENS CALCULATION

Multiple use cases were tested to see if the users were receiving the right amount of tokens:

Alphas owned	Betas owned	Gamma owned	Amount received	Notes
1	0	0	6,52005E+21	
0	1	0	1,41855E+21	
0	0	1	0 (Nothing to claim)	
1	1	0	7,93861E+21	
1	0	1	7,18968E+21	-> 6520054000000000000000 + 669627000000000000000
0	1	1	2,08818E+21	-> 1418552000000000000000 + 669627000000000000000
2	0	1	1,37097E+22	-> (6520054000000000000000 x 2) + 669627000000000000000
2	0	2	1,43794E+22	-> (6520054000000000000000 x 2) + (669627000000000000000 x 2)
1	0	3	7,18968E+21	-> 6520054000000000000000 + 669627000000000000000

The calculation of the token amounts that can be claimed are correct.

The gamma tokens owned will only increase the amount of tokens that can be claimed in the case that the gamma token can be paired with another alpha/beta token owned, as it is specified in the line 167 of the contract: `uint256 gammaToBeClaim = min(unclaimedAlphaBalance + unclaimedBetaBalance , unclaimedGamaBalance);`

This can also be seen in the last line of the table shown above.

4.2 CONTRACT TOKEN BALANCE

100000000_000000000000000000 tokens will be distributed in this airdrop. If every NFT is claimed with the maximum bonus possible, 100000000 _000000000000000000 as the balance of the contract would be enough?

Amount of alpha tokens = 10000
Amount of beta tokens = 17931
Amount of gamma tokens = 9602

In the case that each of those NFTs are claimed, this is the amount of tokens airdropped by the contract:

(10000 x 6520_054000000000000000)+ (17931 x 1418_552000000000000000)+ (9602 x 669_627000000000000000)= 97066354_366000000000000000

As 97066354_366000000000000000 is lower than 100000000_000000000000000000 the contract will always have enough tokens to distribute in the claims.

4.3 CLAIMING THE SAME TOKEN ID TWICE

As can be seen below, it is not possible to claim the same tokenId twice.

In the first example, the user had just an alpha NFT. In the first call to claimTokens() he received 6520.054 tokens. In the second call, as the tokenId he was holding was already claimed, he did not receive any token.

In the second example, the user had, once again, only an alpha NFT. He called claimTokens() receiving 6520.054 tokens, and then he transferred the NFT to another account. From this new account, he called claimTokens () but as this tokenId was already claimed from the other account, he did not receive any token.

From the same account:

```
contract_alpha.balanceOf(user1) -> 1
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 0
contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 100000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x566ba59c8401b4b0b71e89254fbf6ade05ae2551bafc072571fa5904842ae272
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14134461   Gas used: 145963 (0.02%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 99993479946000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 6520054000000000000000


Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x964c1db863046ace35fe7dc272f0b85f28ace26c6043ca9d4a2add09882a559a
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 1
  AirdropGrapesToken.claimTokens confirmed   Block: 14134462   Gas used: 80576 (0.01%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 99993479946000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 6520054000000000000000
```

From another account:

```
contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 1000000000000000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': user1})
Transaction sent: 0x5622a47fe02680fc81ae48ef422a95f3d33e414babb47efae0fc3ae0a23f7415
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14134582   Gas used: 145963 (0.02%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 999934799460000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 65200540000000000000000


Calling -> contract_alpha.transferFrom(user1.address, newacc.address, 1, {'from': user1})
Transaction sent: 0x3e4fff04e6ac7b69684924766cc363b4023bdd8457fbf9033208a06cc9c9a885
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 1
  SimpleERC721.transferFrom confirmed   Block: 14134583   Gas used: 51784 (0.01%)


contract_alpha.balanceOf(user1) -> 0
contract_beta.balanceOf(user1) -> 0
contract_gamma.balanceOf(user1) -> 0
contract_alpha.balanceOf(newacc) -> 1
contract_beta.balanceOf(newacc) -> 0
contract_gamma.balanceOf(newacc) -> 0
Calling -> contract_AirdropGrapesToken.claimTokens({'from': newacc})
Transaction sent: 0x02718456619dcc0588f34e2892cb795595e61a19241693f219e59a608ffc4407
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  AirdropGrapesToken.claimTokens confirmed   Block: 14134584   Gas used: 80576 (0.01%)

contract_ERC20_SimpleToken.balanceOf(contract_AirdropGrapesToken) -> 999934799460000000000000000
contract_ERC20_SimpleToken.balanceOf(user1) -> 65200540000000000000000
contract_ERC20_SimpleToken.balanceOf(newacc) -> 0
```

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

AirdropGrapesToken.sol

```
AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).unclaimedGamaBalance (contracts/AirdropGrapesToken.sol#159) is a local variable never initialized
AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).unclaimedAlphaBalance (contracts/AirdropGrapesToken.sol#145) is a local variable never initialized
AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).unclaimedBetaBalance (contracts/AirdropGrapesToken.sol#152) is a local variable never initialized
AirdropGrapesToken.claimTokens().currentGammaClaimed (contracts/AirdropGrapesToken.sol#121) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#382-403) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modul
es/@openzeppelin/contracts/token/ERC721/ERC721.sol#389-399)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: i < alpha.balanceOf(msg.sender) (contracts/AirdropGrapesToken.sol#105)
AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: tokenId = alpha.tokenOfOwnerByIndex(msg.sender,i) (contracts/AirdropGrapesToken.sol#106)
AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: i_scope_0 < beta.balanceOf(msg.sender) (contracts/AirdropGrapesToken.sol#113)
AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: tokenId_scope_1 = beta.tokenOfOwnerByIndex(msg.sender,i_scope_0) (contracts/AirdropGrapesToken.sol#114)
AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: i_scope_2 < gamma.balanceOf(msg.sender) (contracts/AirdropGrapesToken.sol#122)
AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) has external calls inside a loop: tokenId_scope_3 = gamma.tokenOfOwnerByIndex(msg.sender,i_scope_2) (contracts/AirdropGrapesToken.sol#123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#389)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#382-403) potentially used before declaration: retval == IERC721Receiver.onERC721Received.selector (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#390)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#391)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#382-403) potentially used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#392)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#391)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#382-403) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#396)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135):
        External calls:
        - grapesToken.safeTransfer(msg.sender,tokensToClaim) (contracts/AirdropGrapesToken.sol#131)
        State variables written after the call(s):
        - totalClaimed += tokensToClaim (contracts/AirdropGrapesToken.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135):
        External calls:
        - grapesToken.safeTransfer(msg.sender,tokensToClaim) (contracts/AirdropGrapesToken.sol#131)
        Event emitted after the call(s):
        - AirDrop(msg.sender,tokensToClaim,block.timestamp) (contracts/AirdropGrapesToken.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

AirdropGrapesToken.claimTokens() (contracts/AirdropGrapesToken.sol#96-135) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= claimStartTime && block.timestamp < claimStartTime + claimDuration,Claimable period is finished) (contracts/AirdropGrapesToken.sol#97)
AirdropGrapesToken.claimUnclaimedTokens() (contracts/AirdropGrapesToken.sol#182-190) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > claimStartTime + claimDuration,Claimable period is not finished yet) (contracts/AirdropGrapesToken.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#382-403) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#395-397)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#27-37) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33-35)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#196-216) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#208-211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.10', '^0.8.0']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4)

        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - 0.8.10 (contracts/AirdropGrapesToken.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#80-82) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#109-115) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#169-171) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#179-188) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#142-144) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#152-161) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC721._burn(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#302-314) is never used and should be removed
ERC721._mint(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#280-290) is never used and should be removed
ERC721._safeMint(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#248-250) is never used and should be removed
ERC721._safeMint(address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#256-266) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#40-51) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#56-66) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version0.8.10 (contracts/AirdropGrapesToken.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#55-60):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#58)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#123-134):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#132)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#152-161):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#159)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#179-188):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#186)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#179) is not in mixedCase
Parameter AirdropGrapesToken.startClaimablePeriod(uint256)._claimDuration (contracts/AirdropGrapesToken.sol#81) is not in mixedCase
Parameter AirdropGrapesToken.getClaimableTokenAmount(address)._account (contracts/AirdropGrapesToken.sol#137) is not in mixedCase
Parameter AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address)._account (contracts/AirdropGrapesToken.sol#143) is not in mixedCase
Variable AirdropGrapesToken.ALPHA_DISTRIBUTION_AMOUNT (contracts/AirdropGrapesToken.sol#18) is not in mixedCase
Variable AirdropGrapesToken.BETA_DISTRIBUTION_AMOUNT (contracts/AirdropGrapesToken.sol#19) is not in mixedCase
Variable AirdropGrapesToken.GAMMA_DISTRIBUTION_AMOUNT (contracts/AirdropGrapesToken.sol#20) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).tokenId_scope_1 (contracts/AirdropGrapesToken.sol#154) is too similar to AirdropGrapesToken.claimTokens().tokenId_scope_3 (contracts/AirdropGrapesToken.sol#123)
Variable AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).tokenId_scope_1 (contracts/AirdropGrapesToken.sol#154) is too similar to AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).tokenId_scope_3 (cont
racts/AirdropGrapesToken.sol#161)
Variable AirdropGrapesToken.claimTokens().tokenId_scope_1 (contracts/AirdropGrapesToken.sol#114) is too similar to AirdropGrapesToken.claimTokens().tokenId_scope_3 (contracts/AirdropGrapesToken.sol#123)
Variable AirdropGrapesToken.claimTokens().tokenId_scope_1 (contracts/AirdropGrapesToken.sol#114) is too similar to AirdropGrapesToken.getClaimableTokenAmountAndGammaToClaim(address).tokenId_scope_3 (contracts/AirdropGrapesToken.sol#161)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-56)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#62-65)
name() should be declared external:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81)
symbol() should be declared external:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#112-122)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#136-138)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#150-159)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721.safeTransferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#164-170)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#37-40)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#52-55)
getClaimableTokenAmount(address) should be declared external:
        - AirdropGrapesToken.getClaimableTokenAmount(address) (contracts/AirdropGrapesToken.sol#137-141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- The Reentrancies flagged by Slither are false positives. The functions correctly follow the check, effects, interactions pattern, mitigating any Reentrancy risk.

AUTOMATED TESTING

THANK YOU FOR CHOOSING

HALBORN