Phore Smart Contract Final Audit Report

Project Synopsis

Project Name	**Phore**
Platform	Ethereum, Solidity
Github Repo	Not Provided
Deployed Contract	Not Deployed
Total Duration	5 Days
Timeline of Audit	**25th June 2021 to 28th June 2021**

Contract Details

Total Contract(s)	2
Name of Contract(s)	PHPHolders , Airdrop.sol
Language	Solidity
Commit Hash	`fe27151d7f5cefee40f60d51e7b5c9abdd117b14`

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	0	0
Medium Severity	0	2
Low Severity	1	4
Information	0	0
Total Found	1	6

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: Airdrop

Medium Severity Issues

A.1 Require Statement Validation contains a tautology or contradiction

Status: CLOSED

Contract - EMIVamp
SWC Reference : SWC 123 – Requirement Violation
CWE Reference : CWE 345 - Insufficient Verification of Data Authenticity

Line no: 43

Explanation:

The ***require*** statement used in the ***claimRestOfTokens*** function of the Airdrop contract might contain some contradictory condition.

- ***claimRestOfTokens at Line 43***

```
42    function claimRestOfTokens() public onlyOwner returns(bool)  {
43        require(_graphene.balanceOf(address(this)) >= 0);
44        require(_graphene.transfer(owner(), _graphene.balanceOf(add
45        return true;
46    }
```

The condition ensures that the value of **balance of Graphene Token** in the contract is either ***greater than or equal to ZERO.***

Is this Intended?

This logic is not adequate as it qualifies the require statement at Line 43 even if the contract doesn't hold any **graphene token balance.** This, in fact, triggers the **transfer** function(*Line 44*) with ZERO amount of tokens transferred to the owner.

Recommended:

If the above-mentioned scenario was not intended, the **require** statement should be modified accordingly.

For instance:

```
function claimRestOfTokens() public onlyOwner returns(bool)  {
  require(_graphene.balanceOf(address(this)) > 0,"Contract HAS ZERO
Graphene Tokens");
```

```
        ...
    }
```

A.2 Loops are extremely Costly

Status: CLOSED

SWC Reference : SWC 128 – DoS With Block Gas Limit
CWE Reference : CWE 400 - Uncontrolled Resource Consumption

Line no - 143

Explanation:

Airdrop contract includes a **for loop** that uses 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 mentioned lines of the contracts mentioned below:

- ***_possessMultiplePhrAddressAndClaimGFN at Line 143***

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 = phrAddresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
        possessPhrAddress(sender, phrAddresses[i], signatures[i]);
        claimGFN(sender, merkleProofs[i], phrAddresses[i], amounts[i]);
    }
```

Low Severity Issues

A.3 Comparison to boolean Constant

Status: CLOSED

SWC Reference : SWC 135 – Code With No Effects
CWE Reference : CWE 1164 - Irrelevant Code
Line no - 96

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.

```
95      function claimGFN(address sender, bytes32[] memory merkleProof, string memory phrAddr
96          require(_spent[phrAddr] != true, "Already spent!");
97          require(amount > 0);
98          _spent[phrAddr] = true;
```

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

A.4 External Visibility should be preferred

Status: CLOSED

SWC Reference : SWC 100 – Function Visibility
CWE Reference : CWE 710 - Improper Adherence to Coding Standards
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 functions should must be marked as **external** within the contract:

- ***setRoot***
- ***setGraphene***

- ***setPhrHolders***
- ***contractTokenBalance***
- ***claimRestOfTokens***
- ***possessPhrAddressAndClaimGFN***
- ***possessMultiplePhrAddressAndClaimGFN***

Recommendation:

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

A.5 Absence of Error messages in Require Statements

Status: Partially CLOSED

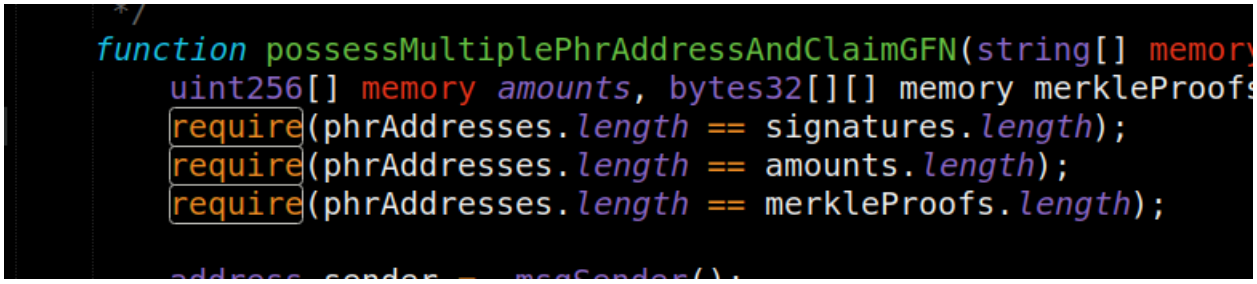
Line no - 44,138,139,140

Note: *The above-mentioned lines still have* ***require statements*** *with no error messages.*

Description:

The **Airdrop** contract includes a few functions(*at the above-mentioned lines*) that don't contain any error message in the ***require*** statement.

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

B. Contract Name: PHRHolders

Low Severity Issues

B.1 External Visibility should be preferred

Status: CLOSED

SWC Reference : SWC 100 – Function Visibility

CWE Reference : CWE 710 - Improper Adherence to Coding Standards

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 functions should must be marked as **external** within the contract:

- ***setWhiteListedSigner***
- ***setSenderAsPhoreOwner***

Recommendation:

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

B.2 Absence of Zero Address Validation

Status: CLOSED

SWC Reference : SWC 123 – Requirement Violation

CWE Reference : CWE 345 - Insufficient Verification of Data Authenticity

Description:

Some of the contracts initialize imperative state variables in the constructor or functions.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented to ensure that no invalid argument is passed while initializing such state variables.

The following functions in lacks a Zero Address Validations at the lines mentioned in specific contracts:

- ***setWhiteListedSigner*** *at Line 25*

```
23      * @notice Change white listed signer.
24      */
25      function setWhiteListedSigner(address newSigner) public onlyOwner
26      {
27          _whiteListedSigner = newSigner;
28      }
29
```

Recommendation:

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

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