SMART CONTRACT AUDIT REPORT

For

W GREEN PAY (Order # 12DEC2018A)

Prepared By: Yogesh Padsala

Prepared on: 12/12/2018

audit@etherauthority.io

Prepared For: W GLOBAL INVESTMENT

https://www.wpay.sg

Table of Content

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2. Overview of the audit

The project has following files:

- lockupWGP.sol

It contains approx **75** lines of Solidity code. All the functions and state variables are **not** well commented using the natspec documentation. However that does not raise any vulnerability.

The audit was performed by Yogesh Padsala, from EtherAuthority Limited. Yogesh has extensive work experience of developing and auditing the smart contracts.

The audit was based on the solidity compiler 0.5.1+commit.c8a2cb62 with optimization enabled compiler in remix.ethereum.org

This audit is for a additional feature implementation, as per document provided.

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Not Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	No Passed
	Other code specification issues	Passed
Gas Optimization	Assert() misuse	Passed
	High consumption ‘for/while’ loop	No Passed
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	Evil mint/burn	Passed
	The maximum limit for mintage not set	Passed
	“Fake Charge” Attack	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed
Auto Fuzzing		Passed

Overall Audit Result: PASSED

3. Attacks tested on the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

3.1: Over and under flows

Althogh SafeMath library is included in the contract, but there is no necessary of using it, as there are no possibility of over and under flows anywhere in both the contracts.

3.2: Short address attack

Although this contract **is not vulnerable** to this attack, it is highly recommended to call functions after checking validity of the address from the outside client.

3.3: Visibility & Delegatecall

Delegatecall is not used in the contract thus it does not have this vulnerability. And visibility is also used properly.

3.4: Reentrancy / TheDAO hack

Use of “require” function and Checks-Effects-Interactions pattern in this smart contract mitigated this vulnerability.

3.5: Forcing ether to a contract

Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability

3.6: Denial Of Service (DOS)

There **is** a process consuming loops in the contracts which can be used for DoS attacks. But that is in the owner only function, which means very less likely owner himself will do DoS to his own contract. Also, there is no progressing state based on external calls, and thus this contract is not prone to DoS.

4. Good things in the smart contract

4.1 Checks-Effects-Interactions pattern

While transferring tokens, this contract does all the process first and then transfers them. The same while doing other process too. This is very good practice which prevents malicious possibility. For example: transfer() function.

4.2 Functions input parameters passed

The functions in this contract verifies the validity of the input parameters, and this validations cannot be by-passed in anyway.

4.3 Good input validations

```
function releaseWgp() external onlyOwner {
    require(recipient != 0x0 && startDate != 0 && now > startDate + lockedPeriod);
    uint256 balance = wgp.balanceOf(address(this));
    wgp.transfer(recipient, balance);
}
```

This function checks whether there is recipient address exist as well as startDate is also added. Since that is updated by the owner, so it is good practice to check that before doing further token transfer.

5. Critical vulnerabilities found in the contract

Critical issues that could damage heavily the integrity of the contract. Some bug that would allow attackers to steal ether is a critical issue.

=> No Critical vulnerabilities found

6. Medium vulnerabilities found in the contract

Those vulnerabilities that could damage the contract but with some kind of limitations. Like a bug allowing people to modify a random variable.

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

Those do not damage the contract, but better to resolve and make code clean.

7.1: Costly Loop

At line #56, the function named, airdropToken() having loop which can go to infinite.

Now, we understand that is owner only function, and owner never do DoS attack to his own contract, but again, it is always better to place some kind of iteration limitations to prevent any unexpected mistakes.

A require function which limits array.length to less than 200 will ideally useful.

7.2: Old compiler version

The code is not compatible with solidity latest version 0.5.1. So, it is highly recommended to make the code compatible to it.

There are so many breaking changes introduced from version 0.5.0:

https://solidity.readthedocs.io/en/v0.5.0/050-breaking-changes.html

7.3: Extra gas consumption

At line #57, recipients.length is used in the for loop. So, while using state variable, .length in the condition of for loop, every iteration of loop consumes extra gas.

So, it is recommended to store that value in a variable and then use it in the loop. For example:

```
uint256 recipientsLength = recipients.length;
for (uint256 i = 0; i < recipientsLength; i++) {
    wgp.transfer(recipients[i], values[i] * 10**18);
}
```

Also, it would be better if token value would be in WEI to save some gas.

8. Discussions and improvements

8.1 Lock period of tokens can be influenced by owner

The function releaseWgp() in WgpHolder contract does prevent the token transfer for the start date + 180 days.

However, this lock period can be influenced by the owner by changing the start date at any time. So, if owner set start date as 1 and so that condition will pass and he can do the token transfer.

8.2 While using SafeMath library

SafeMath library is not used anywhere in both the contracts. So, better to remove the following lines, just to save some little gas:

```
using SafeMath for uint256;
```

8.3 Consider using self-destruct function

It many times happens, where contract owner would need to upgrade the contract or to add any important feature in the contract.

So, the only way that can be possible by creating brand new contract and destroying the old one. And that time, self-destruct comes to help.

9. Summary of the Audit

Overall, the code is simple and straightforward. apart from few improvements suggested above, rest is pretty good.

Please try to check the address and value of token externally before sending to the solidity code.

It is also encouraged to run bug bounty program and let community help to further polish the code to the perfection.