Ether Authority

SMART CONTRACT AUDIT REPORT

For

CarNomic Token (Order #02JUL2019)

Prepared By: Yogesh Padsala

Prepared on: 02/07/2019

audit@etherauthority.io

Prepared For: Carnomic Ltd.

https://www.carnomic.io

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 file:

- https://etherscan.io/address/0x2e0c40beb655a988e087ad71ca191a2806ac55ef#contracts

It contains **114** lines of Solidity code. All the functions and state variables are not well commented, but that does not raise any vulnerability, but it would have raised readability.

The audit was performed by two senior solidity auditors from EtherAuthority. The team has extensive work experience of developing and auditing the smart contracts.

This smart contract reflects correct data according to white paper found at:

https://www.carnomic.io/wp/Carnomic-White-Paper-en.pdf

This audit procedure also included the use of automated software to further scan of the code to identify potential issues:

For example:

https://tool.smartdec.net/scan/24be0ae838eb45178730397 93d9b3cbe

We checked those reports carefully and confirm that some of the warnings, either are just for information purpose or not very critical for our use case!

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version is 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	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Visibility not explicitly declared	Not Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Not Passed
	Other code specification issues	Passed
Gas Optimization	Assert() misuse	Moderated
	High consumption ‘for/while’ loop	N/A
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	The maximum limit for mintage not set	N/A
	“Short Address” Attack	Passed
	“Double Spend” Attack	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 on the code. Some of those are as below:

3.1: Over and under flows

SafeMath library is **not** used in the contract, but proper variable validations prevented the possibility of overflow and underflow attacks.

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 No** any process consuming loops in the contracts which can be used for DoS attacks. and thus this contract is not prone to DoS.

4. Good things in the smart contract

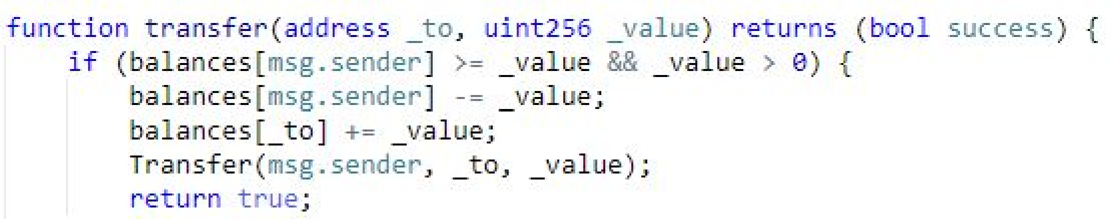
4.1 Checks-Effects-Interactions pattern

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 No unnecessary validations



```
function transfer(address _to, uint256 _value) returns (bool success) {
    if (balances[msg.sender] >= _value && _value > 0) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        Transfer(msg.sender, _to, _value);
        return true;
```

Although use of SafeMath library also would be good programming flow.

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 - Good job team!

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 - Good job again!

7. Low severity vulnerabilities found

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

7.1: Compiler version should be fixed

The contract has lower solidity version than the current one. This version gap is quite high and there were many improvements afterwards.

So, it is good practice to deploy the contract having latest solidity version. The solidity version at a time of audit is: 0.5.10

7.2: Deprecated elements

The way constructor function was defined is deprecated. You need to use “constructor” keyword to define constructor function.

The functions declared as “*constant*” are also deprecated. They need to be declared as *view* or *pure*.

Invoking events without "*emit*" prefix is too deprecated.

7.3: No explicit visibility

Visibility is not specified at line #53, #64, #76, #80, #86, #90, #91, #109. Please note that this is not a big issue as it takes default to "*public*". But it's suggested to explicitly define visibility to avoid confusion.

7.4: No Transfer event in constructor

The constructor function assigns initial supply of tokens to owner. But it does not log for this transaction. It's good to add a Transfer event so it properly log this particular transaction.

7.5: Use *require* instead of *assert* in SafeMath library

If assert check fails, then it will consume all the remaining gas in transaction call. This would give users a surprised high charge in such failed transactions.

So, it's better to use *require*, which only takes gas cost which was used to execute function call up to that point.

8. Gas Optimization Discussion

=> The Contract is most optimum for the gas cost. There is no gas expensive loops, or logical unnecessary processes.

9. Discussions and improvements

9.1 No direct burn function

Whitepaper (page #16) mentioned about token burn. But this contract does not have direct burn function. So, to burn any tokens, users have to send that to zero address (0x0).

9.2 approve() of ERC20 Standard

To prevent attack vectors regarding approve() like the one described here: https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit , clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH the contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

9.3 While using SafeMath library

SafeMath library code is included. But it is not used in contract anywhere. Although we checked that the arithmetic conditions do not cause any underflow or overflow, but if the safemath is not being used then better to remove, or use it in appropriate arithmetic calculations!

9.4 Consider adding ownership contracts

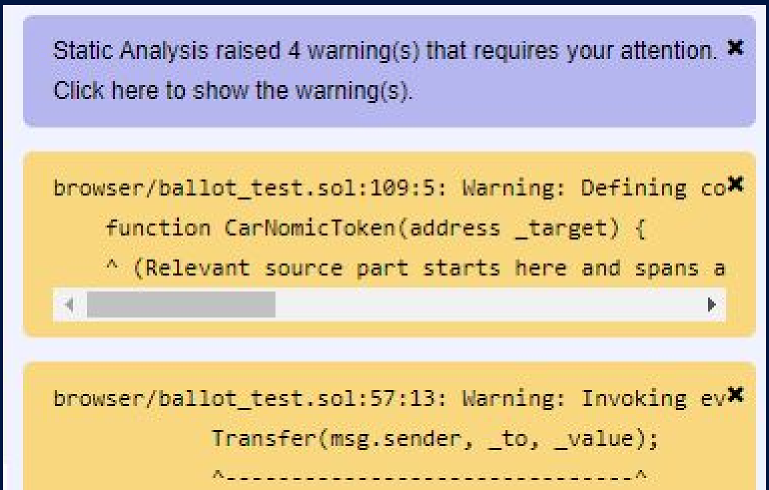
Ideally, the owner of the contract should be defined at the time of contract deployment. And who can do all the administrative functions (if any).

This is useful to manage ownership of the contract down the road.

10. Summary of the Audit

Overall, the code is simple and straightforward ERC20 implementation. apart from few improvements suggested above, rest is pretty good.

Compiler showed couple of warnings, as below:



Static Analysis raised 4 warning(s) that requires your attention.
Click here to show the warning(s).

```
browser/ballot_test.sol:109:5: Warning: Defining co
    function CarNomicToken(address _target) {
    ^ (Relevant source part starts here and spans a
```

```
browser/ballot_test.sol:57:13: Warning: Invoking ev
            Transfer(msg.sender, _to, _value);
            ^-------------------------------^
```

Now, we checked that the warnings in purple division, are due to their static analysis, which includes like gas estimations and all. So, it is important to supply correct gas values while calling various functions.

Those warnings can be safely ignored as should be taken care while calling the smart contract functions.

On another hand, then warnings in purple division should be resolved.

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.