AGATE SMART CONTRACT AUDIT

Prepared for: AGATE
Prepared by: BLOCKETO
13 August 2018
Proposal number: 0.1

DISCLAIMER

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to:
(i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Blocketo and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Blocketo) owe no duty of care towards you or any other person, nor does Blocketo make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Blocketo hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Blocketo hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Blocketo, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

BACKGROUND

BLOCKETO was commissioned by AGATE to perform smart contract code review of their smart contract code on 11th August 2018. The review was conducted between 11th August 2018 and 13th August 2018.
The report is organised into the following sections.

- Executive Summary: A high-level description of the findings of the review.
- Technical analysis: Our detailed analysis of the Smart Contract code
  The information in this report should be used to understand overall code quality, security, correctness and meaning that code will work as Agate describes in the White paper. No Technology stack review was performed. The analysis is entirely limited to Smart contract code.

EXECUTIVE SUMMARY

AGATE is a ERC20 based token. AGATEs smart contracting is built based on Open Zeppelin framework and uses many libraries of Zeppelin Solidity.
In the audit, we have reviewed the smart contract's code that implements the token mechanism.
We have provided independent technical audit to remove smart contract uncertainties and to keep it safe from any serious hacks. Smart contract audit keeps clients protected from any fraudulent acts.
The testing was performed both manually and using automated tools. We have analysed the smart contract code line by line and reported the suspicious code. After manual analysis, we have used automated tools to make sure the coverage is thorough.

Besides, the results of the automated analysis, manual verification was also taken into account. The complete contract was manually analyzed, every logic was checked and compared with the one described in the whitepaper. The manual analysis of code confirms that the Contract does not contain any serious susceptibility. No divergence was found between the logic in Smart Contract and the whitepaper.

This audit is into the technical and security aspects of the AGATE smart contract. The key aim of this audit is to ensure that funds contributed to these contracts are not by far attacked or stolen by any third parties. The next aim of this audit is that ensure the coded algorithms work as expected. The audit of Smart Contract also checks the implementation of token mechanism i.e. the Contract must follow all the ERC20 Standards.

This audit is purely technical and is not an investment advice. The scope of the audit is limited to the following source code files:

- AgateToken
- BasicToken
- BurnableToken
- ERC20
- ERC20Basic
- Owned
- SafeERC20
- SafeMath
- StandardToken
- TokenVesting

Testing

AGATE smart contract went through rigorous testing, which focused on the security features of the smart contract. During the complete Test phase the security of the Project was prime consideration. Major Task was to find and describe the security issues in the Smart Contract.

We have considered the following Standards for the Smart contract code review:

- OWASP Top 10 [Crowd Sale]
- ERC20 [Token Contract]

Primary checks followed during testing of Smart Contract is to see that if code :

- We check the Smart Contracts Logic and compare it with one described in Whitepaper.
- The contract code should follow the Conditions and logic as per user request.
- We deploy the Contract and run the Tests.
- We make sure that Contract does not lose any money/Ether.

Vulnerabilities Check

Smart Contract was scanned for commonly known and more specific vulnerabilities.
Following are the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

TIMESTAMP DEPENDENCE

The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. Block numbers and average block time can be used to estimate time (suggested). AGATE smart contract uses timestamp only to set vesting and it is not critically vulnerable to this type of attack.

GAS LIMIT AND LOOPS

Loops that do not have a fixed number of iterations, hence due to normal operation, the number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. AGATE smart contract is free from the gas limit check as the contract code does not contain any loop in its code.

COMPILER VERSION

Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. AGATE smart contract is locked to a specific compiler version of 0.4.24 which is good coding practice.

ERC20 STANDARDS

AGATE smart contract follows all the universal ERC20 coding standards and implements all its functions and events in the contract code. This standard is followed in the StandardToken.sol file of the AGATE smart contract.

REDUNDANT FALLBACK FUNCTION

The standard execution cost of a fallback function should be less than 2300 gas, AGATE smart contract code is free from this vulnerability.

UNCHECKED MATH

Need to guard uint overflow or security flaws by implementing the proper math logic checks. The AGATE smart contract uses the popular SafeMath library for critical operations to avoid arithmetic over- or underflows and safeguard against unwanted behavior. The SafeMath library is implemented in safemath.sol file. In particular the balances variable is updated using the safemath operation.

EXCEPTION DISORDER

When an exception is thrown, it cannot be caught: the execution stops, the fee is lost. The irregularity in how exceptions are handled may affect the security of contracts.

UNSAFE TYPE INFERENCE

It is not always necessary to explicitly specify the type of a variable, the compiler automatically infers it from the type of the first expression that is assigned to the variable.

REENTRANCY

The reentrancy attack consists of the recursively calling a method to extract ether from a contract if user is not updating the balance of the sender before sending the ether.
In AGATE smart contract calls to external functions happen after any changes to state variables in the contract except once internally so the contract is not vulnerable to a reentrancy exploit.

DOS WITH (UNEXPECTED) THROW

The Contract code can be vulnerable to the Call Depth Attack! So instead, code should have a pull payment system instead of push. The AGATE smart contract securely handles the all the payment related scenario thus it is not vulnerable to this attack.

DOS WITH BLOCK GAS LIMIT

In a contract by paying out to everyone at once, contract risk running into the block gas limit. Each Ethereum block can process a certain maximum amount of computation. If one try to go over that, the transaction will fail. AGATE smart contract implements the push over pull payment to remove the above issue.

EXPLICIT VISIBILITY IN FUNCTIONS AND STATE VARIABLES

Explicit visibility in the function and state variables are provided. Visibility like external, internal, private and public is used and defined properly.

TECHNICAL ANALYSIS

Variables

The following are the variables that define the state of the smart contract.

1. Token Symbol : AGT
2. Token Name : AGATE
3. Hard Cap : 490M
4. Address of Tokens for sale
5. Address of Bounty tokens
6. Address of Team tokens pool
7. Address of Advisors tokens pool
8. Address of Reserve tokens pool
9. has SaleClosed
10. is TradingOpen
11. Date for token vesting

Features in Smart Contract

BASICTOKEN

1. Transferrable - The tokens can be transferred from one entity to other(like to exchanges for trading). This transfer can be made by using any ethereum wallet which supports ERC20 token standard, for eg. MyEtherWallet, Mist Etc.

Code Line: 86-95

```
function transfer(address _to, uint256 _value)
    public
    returns (bool)
{
    require(_to != address(0));
    require(_value <= balances[msg.sender]);
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;
}
```

2. Viewable Tokens - As you already may be aware with the transparent nature of blockchain, all the tokens holders and their exact balance is made clearly visible, on Ethereum blockchain explorers like Etherscan and Ethplorer. Their are functions which are implemented to return informations like token balance of any particular token holder, the token allowance amount of any particular Token holder, which he has allowed to any other entity(Ethereum address).

Code Line: 138-144

```
function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
        }
```

STANDARDTOKEN

1. Approvable - Any Token holder if he wills, can approve some other address, who will on his behalf transfer the approved amount of tokens from token holder's to others.

Code Line: 157-161

```
function approve(address _spender, uint256 _value)public returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
        }
```

AGATETOKEN

1. Burning of Tokens - The leftover tokens are meant to be burnt, if all the 318.5 million Coins are not sold during the AGATE Tokensale process, Thereby decreasing the Total Supply by that amount, this is done only only once the Auction is finalised smart contract.

Code Line: 466 - 471

```
function closeSale()
  external
  onlyOwner
  beforeSaleClosed
  {
     _burn(saleTokensAddress, balances[saleTokensAddress]);
     saleClosed = true;
  }
```

2. Ownable - The smart contract and the tokens implemented it are owned by a particular entity (ethereum public address). To use those tokens the owner will have to sign with his private key. As we deploy the smart contract on Ethereum blockchain, initially all the tokens will be owned by the owner of the smart contract i.e. the entity offering the crowdsale (we walk you through the procedure to deploy the token, so you will own all the tokens before crowdsale starts).

Code Line: 224-227

Modifier:

```
modifier onlyOwner() {
        if (msg.sender != owner) {
        revert();
        }
        _;
        }
```

Security Risks

Potential Violation of Checks-Effects-Interaction pattern in TokenVesting.revoke(contract ERC20Basic): Could potentially lead to re-entrancy vulnerability.
Line : 440

Deprecated use of `this` for address can lead to discrepancy in TokenVesting.revoke(contract TokenVesting).
`this` has been deprecated in favour of `address(this)`
Line : 309

Double Event Issuance in Burnable._burn(contract BurnableToken):
Having two events fire for a single function leads to incorrect logging and runs contrary to ERC-20 guidelines.
Line : 370

SUMMARY OF THE AUDIT

Overall the code is well commented and clear on what it's supposed to do for each function.
The mechanism to distribute tokens is quite simple so it shouldn't bring major issues.
It is recommended to update the libraries to the the latest ones at https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts