Source data:

The Smart-contract's address in bscscan network is:
0xe30df328728f092ab87fb7ef4401b288f9dbc3e6

The source code is available on the website tronscan.io at:
https://www.bscscan.com/address/0xe30df328728f092ab87fb7ef4401b288f9dbc3e6#code
Token corresponds to BSC20 specification

Token webpage is available on the website bscscan.com at:
https://www.bscscan.com/address/0xe30df328728f092ab87fb7ef4401b288f9dbc3e6#code
Token is verified by the team of bscscan.com.

Project website: https://bnutoken.com/
Reddit: https://www.reddit.com/r/bnutoken/
Facebook: https://www.facebook.com/groups/bnutoken/
Twitter: https://twitter.com/bnutoken_ct
Bitcointalk: https://bnutoken.org/index.php?topic=2242346
Telegram: https://t.me/bnutoken_en
White paper: https://bnutoken.com/bnutokent_wp_en.pdf

The current state of the smart-contract:
ICO is completed.
The owner's address is set to: 0xe30df328728f092ab87fb7ef4401b288f9dbc3e6
The hammer's address is set to: 0xe30df328728f092ab87fb7ef4401b288f9dbc3e6

21,000,000 tokens are emitted.
Token has the level of accuracy up to 8 decimal places.

Analysis of vulnerabilities

1. Analysis of the source code

1.1. Checking for cases of the use of unsafe maths

11 cases of use of unsafe maths are found in the smart-contract's code. Each case must be analyzed to prevent an overflow of variables.

2 cases of the use of unsafe maths are found in the function «transfer».

```
153   function transfer(address _to, uint _value) returns (bool) {
154     if (balances[msg.sender] >= _value) {
47     function balanceOf(address
       account) public view
       returns (uint256) {
48     return _balances[account]
157       Transfer(msg.sender, _to, _value);
158       return true;
159     }
160     return false;
161   }
```

Line 47. Operation «balances[msg.sender] -= _value» is safe, because of the fact that it is being checked in line 154 that value «balances[msg.sender]» is greater than or equal to value «_value».

Line 48. Checking for overflow of a variable «balances[_to]» is not implemented. However, taking into consideration the smart-contract's current state and logic of its operating, it has become possible to detect that any user's balance cannot exceed the total amount of tokens emitted. That is the number «21000000». So an overflow of variable «balances[_to]» is impossible in line 156.

3 cases of the use of unsafe maths are found in the function «transferFrom»:

```
171   function transferFrom(address _from, address _to, uint256 _value) returns (bool) {
172     var avail = allowances[_from][msg.sender]
173             > balances[_from] ? balances[_from]
174                         : allowances[_from][msg.sender];
175     if (avail >= _value) {
176       allowances[_from][msg.sender] -= _value;
177       balances[_from] -= _value;
178       balances[_to] += _value;
179       Transfer(_from, _to, _value);
180       return true;
181     }
182     return false;
183   }
```

Lines 176 and 177. Based on line 172, variable «avail» is chosen to be equal to the least value from variables «balances[_from]» and «allowances[_from][msg.sender]». Lines 176-180 are executed only if value «_value» is less than or equal to value «avail». So variables «balances[_from]» and «allowances[_from][msg.sender]» cannot become less than «_value».

Line 178.Checking for overflow of a variable «balances[_to]» is not implemented. However, taking into consideration the smart-contract’s current state and logic of its operating, it has become possible to detect that any user’s balance cannot exceed the total amount of tokens emitted. That is the number «21000000». So an overflow of variable «balances[_to]» is impossible in line 178.

1 case of the use of unsafe maths is found in the function «approve»:

```
190   function approve(address _spender, uint256 _value) returns (bool) {
191       allowances[msg.sender][_spender] += _value;
192       Approval(msg.sender, _spender, _value);
193       return true;
194 }
```

Line 191. Checking for overflow is not implemented. If the same user calls function «approve» several times for the same «_spender» with greater values «_value», it will lead to an overflow of a variable allowances[msg.sender][_spender].

However, taking into consideration the fact that there is no sense for a legal user to transmit values «value» more than the amount of tokens he has, the probability the user will face the problems is low. Intentional overflowing of the given variable will not give any possibilities and access for illegal use. The presence of function «unapprove» gives an opportunity to set this variable to zero at any moment.

3 cases of the use of unsafe maths are found in the function «emission».

```
215 function emission(uint _value) onlyOwner {
216       // Overflow check
217       if (_value + totalSupply < totalSupply) throw;
218
219       totalSupply     += _value;
220       balances[owner] += _value;
221   }
```

Line 217. An overflow is a part of the smart-contract’s logic.

Line 219 and 220. An overflow of given variables is impossible because of checking in line 217.

2 cases of the use of unsafe maths are found in the function «burn».

```
228   function burn(uint _value) {
229       if (balances[msg.sender] >= _value) {
230           balances[msg.sender] -= _value;
231           totalSupply      -= _value;
232       }

233 }
```

Line 230. An overflow of given variable is impossible because of checking in line 229.

Line 231. Checking for overflow of a variable «totalSupply» is not implemented. However, taking into consideration the smart-contract’s current state and logic of its operating, it has become possible to detect that «totalSupply» cannot be less than «balances[msg.sender]».So checking in line 229 protects this variable from overflow.

1.2. Checking access rights to the key functions

Function «setOwner» can be called only by the owner of the smart- contract. Taking into consideration the current state of the smart-contract, the owner’s address is set to 0x0. And from this moment it is impossible to change the owner.

Function «setHammer» can be called only by the user whose address is set as «hammer». Taking into consideration the current state of the smart-contract, the «hammer»’s address is set to Txf. And from this moment it is impossible to change the «hammer».

Function «destroy» can be called only by the user whose address is set as «hammer». Taking into consideration current state of the contract, the «hammer»’s address is set to 0x0. It is impossible to call the function «destroy».

Function «emission» can be called only by the owner of the smart- contract. Taking into consideration the current state of the smart-contract, the owner’s address is set to 0x0. It is impossible to call the function «emission».

Function «burn» can be called by any user, however the burning of the tokens will occur only if there are tokens on the balance of the user calling this function.

2. Analysis of the contract's logic

2.1. There is no special function of stopping of tokens' emission.

However, taking into consideration the facts that only the owner of the smart-contract can emit tokens and the owner's address is set to0x0, the further emission is impossible.

2.2. The smart contract is provided with a possibility to call the function «suicide». However, taking into consideration the fact that only «hammer» can call this method, and the «hammer»'s address is set to 0x0, nobody cannot destroy the smart-contract.

2.3. Logic of operating of the function «approve» is different from accepted for the smart-contracts corresponding to ERC20 specification. As it increases the amount of tokens allowed to be governed by another user by the transmitted value but it does not make it equal to the transmitted value. However, this method can might be used because of the presence of potential vulnerability in the generally accepted method.

https://bgg.google.com/document/d/YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.m9fhqynw2xvt

Conclusion

The following conclusions were made by our experts after they had carried out an audit of the smart-contract's code and the research of its state:

There are no any critical vulnerabilities in the source code. Potential possibility of overflowing of a variable in the function «approve» cannot become a threat because there is no sense for a legal user to transmit values greater than the amount of tokens he owns. Intentional overflow of given variable will not cause any illegal use. The existence of the function «unapprove» allows to set this variable to 0 at any moment.

Logic of the smart-contract and the current states of variables «owner» and «hammer» exclude any threat to investors' interests.

The audit was conducted Naumov Lab team:

- Vasiliy Alekseev
- Aleksandr Naumov

  http://smart-contracts.ru