Contract Audit Report

Prepared by: Roman Golovay @ ICO Experts Agency

Functional Specification for gangFY (GNGFY) TOKEN

– Token will have 18 decimal places.

– Ability to CREATE X number of tokens as and when required. X can be any integer between 1000 to 10,000,000

– Ability to automatically ISSUE tokens upon receiving ETHER. (GangTokenSale Contract)

– Ability to set and adjust the conversion RATE between ETHER and gangFY TOKEN as and when required (1 to 20,000 gangFY Tokens per 1 ETHER), for the operation mentioned in the item (3) above.

– Ability to prevent the creation of new tokens or transfer of tokens to the wallet mentioned in the item (3) above. (pause)

– Ability to TRANSFER the ownership of the contract. (Token contract ownership will be transferred to a multi signature wallet)

– Max supply of 2 000 000 000 (2 billion) tokens

– Ability to BURN gangFY tokens, as and when required.

– OTHER TASKS (We expect from the auditor):

– Verify that the SAFETY features have been implemented properly.

– Find gaps in SAFETY features and make suggestions to improve.

– Use of wallet.gnosis.pm/#/wallets as our multi signature wallet. Any safety concerns?

– Any other Suggestions

IMPORTANT:

We are not going to have an ICO.

1. Ability to CREATE X number of tokens as and when required. X can be any integer between 1000 to 10,000,000

Any amount of tokens can now be additionally created, no limitations.

2. Ability to automatically ISSUE tokens upon receiving ETHER. (GangTokenSale Contract) Implemented.

Function that is currently used to buy tokens is 'transfer', which can be blocked. I strongly recommend to use other function that is allowing to sale tokens with 'transfer' disabled.

3. Ability to set and adjust the conversion RATE between ETHER and gangFY TOKEN as and when required (1 to 20,000 gangFY Tokens per 1 ETHER), for the operation mentioned in the item (3) above.

Limits are disabled when changing ratio. (from 0 to 2^256-1)

4. **Standart contracts are edited, they should to be equal as standard offers**

5. Ability to prevent the creation of new tokens or transfer of tokens to the wallet mentioned in the item (3) above. (pause)

Implemented.

6. Ability to TRANSFER the ownership of the contract. (Token contract ownership will be transferred to a multi signature wallet)

Contract has to know how to cooperate with ICO contracts. I doubt that any third party multi signature wallet knows how to do that. Usualy each ICO requires unique multi signature wallet
This functional will fit if wallet is going to be used for ethereum storage, not a chance it'll be suitable for an owner's wallet.

7. Max supply of 2 000 000 000 (2 billion) tokens

Implemeted as parameter in constructor. I don't know why, but that's how it is. When contract is created you'll have to type 2000000000000000000000000000.
Nicely done by developer, now have to not make a single mistake in all those zeros.

8. Ability to BURN gangFY tokens, as and when required.

```
function burn(uint256 _value) public {
      _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
```

Should be:

```
function burn(uint256 _value) public {
   balances[msg.sender] = balances[msg.sender].sub(_value);
   totalSupply_ = totalSupply_.sub(_value);
   emit Burn(msg.sender, _value);
   emit Transfer(msg.sender, address(0), _value);
}
```

This line is not neccessary:

```
require(_value <= balances[_who]);
```

because look at function "sub"

```
 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
   assert(b <= a);
   return a - b;
 }
```

9.

```
 /**
  * @dev called by the owner to pause, triggers stopped state
  */
 function pause() onlyOwner whenNotPaused public {
   paused = true;
   emit Pause();
 }
 /**
  * @dev called by the owner to unpause, returns to normal state
  */
 function unpause() onlyOwner whenPaused public {
   paused = false;
   emit Unpause();
 }
```

Because of this function your token wount be listed at any token exchange. Limits has to be set.

10. **Inside of crowdsale contract the only token function that is used is ("transfer"), but all the methods are implemented.**

This is how it was done:

```
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: contracts/zeppelin/token/ERC20/ERC20.sol

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

How it should be:

```
contract ERC20 {
  function transfer(address to, uint256 value) public returns (bool);
}
```

Works the same.

11. Again lot of unnecessary code for a simple things.

```
/**
 * @dev low level token purchase ***DO NOT OVERRIDE***
 * @param _beneficiary Address performing the token purchase
 */
function buyTokens(address _beneficiary) public payable {

  uint256 weiAmount = msg.value;
  _preValidatePurchase(_beneficiary, weiAmount);

  // calculate token amount to be created
  uint256 tokens = _getTokenAmount(weiAmount);

  // update state
  weiRaised = weiRaised.add(weiAmount);

  _processPurchase(_beneficiary, tokens);
  emit TokenPurchase(msg.sender, _beneficiary, weiAmount, tokens);

  _forwardFunds();
}

// -----------------------------------------
// Internal interface (extensible)
// -----------------------------------------

/**
 * @dev Validation of an incoming purchase. Use require statements to revert state when conditions are not met. Use super to
concatenate validations.
 * @param _beneficiary Address performing the token purchase
 * @param _weiAmount Value in wei involved in the purchase
 */
function _preValidatePurchase(address _beneficiary, uint256 _weiAmount) internal pure {
  require(_beneficiary != address(0));
  require(_weiAmount != 0);
}

/**
 * @dev Source of tokens. Override this method to modify the way in which the crowdsale ultimately gets and sends its tokens.
 * @param _beneficiary Address performing the token purchase
 * @param _tokenAmount Number of tokens to be emitted
 */
function _deliverTokens(address _beneficiary, uint256 _tokenAmount) internal {
  token.transfer(_beneficiary, _tokenAmount);
}

/**
 * @dev Executed when a purchase has been validated and is ready to be executed. Not necessarily emits/sends tokens.
 * @param _beneficiary Address receiving the tokens
 * @param _tokenAmount Number of tokens to be purchased
 */
function _processPurchase(address _beneficiary, uint256 _tokenAmount) internal {
  _deliverTokens(_beneficiary, _tokenAmount);
}
```

```
function _processPurchase(address _beneficiary, uint256 _tokenAmount) internal {
  _deliverTokens(_beneficiary, _tokenAmount);
}

/**
 * @dev Override to extend the way in which ether is converted to tokens.
 * @param _weiAmount Value in wei to be converted into tokens
 * @return Number of tokens that can be purchased with the specified _weiAmount
 */
function _getTokenAmount(uint256 _weiAmount) internal view returns (uint256) {
  return _weiAmount.mul(rate);
}

/**
 * @dev Determines how ETH is stored/forwarded on purchases.
 */
function _forwardFunds() internal {
  wallet.transfer(msg.value);
}
```

Redone. Does the same:

```
/**
 * @dev low level token purchase ***DO NOT OVERRIDE***
 * @param _beneficiary Address performing the token purchase
 */
function buyTokens(address _beneficiary) public payable {

  uint256 weiAmount = msg.value;
  require(_beneficiary != address(0));
  require(_weiAmount != 0);

  // calculate token amount to be created
  uint256 tokens = _weiAmount.mul(rate);

  // update state
  weiRaised = weiRaised.add(weiAmount);

  token.transfer(_beneficiary, _tokenAmount);
  emit TokenPurchase(msg.sender, _beneficiary, weiAmount, tokens);

   wallet.transfer(msg.value);
```

12. **In the token's contract all constructors are done for compilator 0.4.22 (constructor(){}), and in crowdsale under the compilators of a previous versions (function ContractName(){}). This has to be equal.**

In total:

1. Contacts contain alot of unnecessary code that does literaly nothing. Current contract code can optimized to contain at least 50% less lines, improving gas afterall.

2. Contract isn't working as it was declared in teqnical specification.