'YANG' Token Contract: Review

**Overview**

The project has 5 contracts which contains 223 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

*Nice Features:*

* The contract provides a good suite of functionality that will be useful for the entire contract: It uses [SafeMath](https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol) library to check for overflows and underflows which is a pretty good practise.
* It has ERC20Interface which implements a good architecture.
* The permissions are handled by Owned.sol contract.

**Attacks made to the contract:**

In order to check for the security of the given contract we tested the following default security checks.

**Over and under flows:** An overflow happens when the limit of the type variable uint256, 2\*\*256 is exceeded. Whats happens is the value resets to zero instead of incrementing more.

For instance, if I want to assign a value to a uint bigger than 2 \*\* 256 it will simple go to 0 — this is dangerous.

On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.For example, if you subtract 0 - 1 the result will be = 2 \*\* 256 instead of -1.

This is quite dangerous. However This contract checks for overflows and underflows in [SafeMath](https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol) and there is no instance of direct arithmetic operations.

**Short address attack:** This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

A user creates an ethereum wallet with a traling 0, which is not hard because it’s only a digit. For instance: 0xiofa8d97756as7df5sd8f75g8675ds8gsdg0

Then he buys tokens by removing the last zero:

Buy 1000 tokens from account 0xiofa8d97756as7df5sd8f75g8675ds8gsdg

If the token contract has enough amount of tokens and the buy function doesn’t check the length of the address of the sender, the Ethereum’s virtual machine will just add zeroes to the transaction until the address is complete.

The virtual machine will return 256000 for each 1000 tokens bought. This is a bug of the virtual machine that’s yet not fixed so whenever you want to buy tokens make sure to check the length of the address.

Yang contract does not handle this attack. Whenever address is taken as input parameter its a good practice to check for short address attack.

modifier onlyPayloadSize(uint size) {   
 assert(msg.data.length >= size + 4);   
 \_;   
}   
function transfer(address \_to, uint256 \_value) onlyPayloadSize(2 \* 32) {   
 // do stuff   
}

Sanity Check :

function transfer(address \_to, uint256 \_value) onlyPayloadSize(2 \* 32) {   
 require(\_to != 0, “Not a valid address”)  
}

**Line by Line Comments:**

* Line 103 & 104: string can be replaced by bytes32 to save gas consumption.
* Line 129: instead of balances[address(0)] it can be balances[0x25D769a1b3bcAF7216b0cea17C056130fe1720cf] as in Line 120. This removes inconsistency.
* Line 147 before executing transfer function it is good to check the balance of the msg.sender and throw an error by using require

**require(balances[msg.sender**] **>=** **\_tokens);**

* Line 179 in transferFrom function before we subtract the tokens we are not checking the allowed balance. Though it might fail in Line 180 it’s a good practice to do a sanity check. Eg:

**function transferFrom(address \_from, address \_to, uint256 \_tokens) public returns (bool success) {**

**uint256 allowance = allowed[\_from][msg.sender];**

**require(balances[\_from] >= \_tokens&& allowance >= \_tokens);**

**balances[from] = safeSub(balances[from], tokens)**

**balances[to] = safeAdd(balances[to], tokens);**

**allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);**

**emit Transfer(\_from, \_to, \_value);**

**return true;**

**}**

* Line 212, If the fallback function should not accept ETH then why don’t we remove paybale from the function ?

**Critical vulnerabilities found in the contract**

There aren’t critical issues in the smart contract audited.

**Medium vulnerabilities found in the contract**

This token contract doesn't prevent Short Address Attacks

**Our Recommendation:**

* The compiler version can be upgraded to latest version 0.4.23 as some of the functionalities might be deprecated in future. For example -> use of pure/view instead of constant
* Follow the best practice of using require for condition check.

**Summary of the audit**

Overall the code is well commented and clear on what it’s supposed to do for each function. The mechanism is quite simple so it shouldn’t bring major issues. This is a secure contract that will work as expected