AnChain Contract Audit Repo

Time: 30.7 seconds

Environment: EVM version 1.8.15 | Solidity

version 0.4.24

Number of Lines: 320

Code Classes: AgroLyte, SafeMath

Code Quality

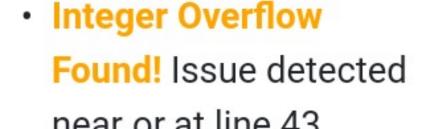


AgroLyte 89.0%

SafeMath 100.0%

AgroLyte

These vulnerabilities were found:





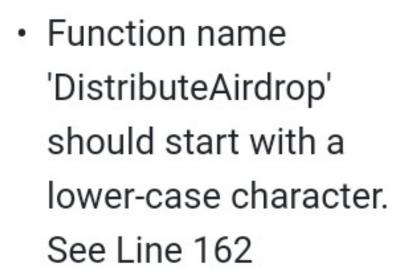
ficul of at fille to

Transaction-Ordering
 Dependence Found!
 Issue detected near or at lines 291, 296

Recommendations

- 'constant' is deprecated. Consider using 'view' instead. See Lines: 50, 56, 62, 238, 278, 282
- Consider using exact language version instead. See Line 1
- Integer division should be used with caution.
 See Lines: 86, 183, 184
- Function name
 'Distribute' should start
 with a lower-case





- Function name

 'DistributeAirdropMultiple should start with a lower-case character.

 See Line 166
- Avoid using 'now' in control expression, since it could be controled by miners.
 See Lines: 190, 190, 190, 198, 198, 198, 220, 220, 220
- This ERC20 function returns false, which may not be correctly.



handled by the caller.
See Line 271

AgroLyte

These vulnerabilities were not found:



- Integer Underflow
- Parity Multisig Bug
- Callstack Depth Attack
- Timestamp
 Dependency
- Re-Entrancy

SafeMath

These vulnerabilities were not found:

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug





- Callstack Depth Attack
- Transaction-Ordering Dependence
- Timestamp
 Dependency
- Re-Entrancy

pragma solidity ^0.4.24;

```
/**
```

- * @title SafeMath
- * @dev Math operations with safety checks that throw on error

*/

library SafeMath {

```
**/
* @dev Multiplies two numbers, throws
on overflow.
*/
function mul(uint256 a, uint256 b)
internal pure returns (uint256 c) {
if (a == 0) {
return 0;
c = a * b;
assert(c / a == b);
return c;
 **
* @dev Integer division of two
numbers, truncating the quotient.
```

*/

```
function div(uint256 a, uint256 b)
internal pure returns (uint256) {
  as#ert(b > 0); // Solidity automatically
throws when dividing by 0
  uin/t/256 c = a / b;
  as#ert(a == b * c + a % b); // There is
no case in which this doesn't hold
return a / b;
 **
* @dev Subtracts two numbers, throws
on overflow (i.e. if subtrahend is
greater than minuend).
*/
function sub(uint256 a, uint256 b)
internal pure returns (uint256) {
```

```
assert(b <= a);
return a - b;
 **
* @dev Adds two numbers, throws on
overflow.
*/
function add(uint256 a, uint256 b)
internal pure returns (uint256 c) {
c = a + b;
assert(c >= a);
return c;
contract ForeignToken {
function balanceOf(address _owner)
```

```
constant public returns (uint256);
function transfer(address _to, uint256
_value) public returns (bool);
}
```

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

contract ERC20 is ERC20Basic {
function allowance(address owner,
address spender) public constant

```
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);
```

contract AgroLyte is ERC20 {

using SafeMath for uint256; address owner = msg.sender;

mapping (address => uint256)

```
mapping (address => mapping
(address => uint256)) allowed;
mapping (address => bool) public
Claimed;
```

parances,

```
string public constant name =
"AgroLyte Token";
string public constant symbol = "AGR";
uint public constant decimals = 8;
uint public deadline = now + 37 * 1
days;
uint public round2 = now + 32 * 1 days;
uint public round1 = now + 17 * 1 days;
```

uint256 public totalSupply = 2100000000000e8; uint256 public totalDistributed;

uint256 public constant requestMinimum = 1 ether / 100; // 0.01 Ether uint256 public tokensPerEth = 5000000e8;

uint public target0drop = 2500; uint public progress0drop = 0;

event Transfer(address indexed _from, address indexed _to, uint256 _value); event Approval(address indexed _owner, address indexed _spender, uint256 _value);

event Distr(address indexed to, uint256 amount);

```
event DistrFinished();
event Airdrop(address indexed _owner,
uint _amount, uint _balance);
event TokensPerEthUpdated(uint
_tokensPerEth);
event Burn(address indexed burner,
uint256 value);
event Add(uint256 value);
bool public distributionFinished = false;
```

modifier canDistr() {
require(!distributionFinished);

```
modifier onlyOwner() {
require(msg.sender == owner);
constructor() public {
uint256 companyFund =
13230000000e8;
owner = msg.sender;
distr(owner, companyFund);
```

function transferOwnership(address newOwner) onlyOwner public { if (newOwner != address(0)) {

```
owner = newOwner;
function finishDistribution() onlyOwner
canDistr public returns (bool) {
distributionFinished = true;
emit DistrFinished();
return true;
function distr(address _to, uint256
_amount) canDistr private returns
(bool) {
totalDistributed =
totalDistributed.add(_amount);
balances[_to] =
balances[_to].add(_amount);
```

```
emit Distr(_to, _amount);
emit Transfer(address(0), _to,
_amount);
return true;
function Distribute(address
_participant, uint _amount) onlyOwner
internal {
require( _amount > 0 );
require(totalDistributed < totalSupply);
balances[_participant] =
balances[_participant].add(_amount);
totalDistributed =
totalDistributed.add(_amount);
```

```
if (totalDistributed >= totalSupply) {
distributionFinished = true;
    }
  log/
emit Airdrop(_participant, _amount,
balances[_participant]);
emit Transfer(address(0), _participant,
_amount);
  }
function DistributeAirdrop(address
_participant, uint _amount) onlyOwner
external {
Distribute(_participant, _amount);
```

```
function
DistributeAirdropMultiple(address[]
_addresses, uint _amount) onlyOwner
external {
for (uint i = 0; i < _addresses.length;
i++) Distribute(_addresses[i], _amount);
  }
function updateTokensPerEth(uint
_tokensPerEth) public onlyOwner {
tokensPerEth = _tokensPerEth;
emit
TokensPerEthUpdated(_tokensPerEth);
  }
function () external payable {
getTokens();
```

```
function getTokens() payable canDistr
public {
uint256 tokens = 0;
uint256 bonus = 0;
uint256 countbonus = 0;
uint256 bonusCond1 = 1 ether / 10;
uint256 bonusCond2 = 1 ether / 2;
uint256 bonusCond3 = 1 ether;
tokens =
tokensererEth.mul(msg.value) /
address investor = msg.sender;
if (msg.value >= requestMinimum &&
now < deadline && now < round1 &&
now < round2) {
```

if(msg value >= bonusCond1 &&

```
msg.value < bonusCond2){
countbonus = tokens * 5 / 100;
else if{msg.value >= bonusCond2 &&
msg.value < bonusCond3){
countbonus = tokens * 10 / 100;
else if{msg.value >= bonusCond3){
countbonus = tokens * 20 / 100;
else if (msg.value >= requestMinimum
&& now < deadline && now > round1 &&
now < round2){
if(msg.value >= bonusCond2 &&
msg.value < bonusCond3){
countbonus = tokens * 5 / 100;
else if{msg.value >= bonusCond3){
countbonus = tokens * 10 / 100;
```

```
elsex
countbonus = 0;
bonus = tokens + countbonus;
if (tokens == 0) {
uint256 valdrop = 500e8;
if (Claimed[investor] == false &&
progress0drop <= target0drop ) {</pre>
distr(investor, valdrop);
Claimed[investor] = true;
progress0drop++;
else{ }
require( msg.value >=
requestMinimum);
else}if(tokens > 0 && msg.value >=
```

```
requestMinimum){
if( now >= deadline && now >= round1
&& now < round2){
distr(investor, tokens);
else{ }
if(msg.value >= bonusCond1){
distr(investor, bonus);
else{ }
distr(investor, tokens);
else∦
require( msg.value >=
requestMinimum);
```

if (totalDistributed >= totalSupply) {
distributionFinished = true:

```
function balanceOf(address _owner)
constant public returns (uint256) {
return balances[_owner];
modifier onlyPayloadSize(uint size) {
assert(msg.data.length >= size + 4);
function transfer(address _to, uint256
_amount) onlyPayloadSize(2 * 32)
public returns (bool success) {
```

```
require(_amount <=
balances[msg.sender]);
balances[msg.sender] =
balances[msg.sender].sub(_amount);
balances[_to] =
balances[_to].add(_amount);
emit Transfer(msg.sender, _to,
_amount);
return true;
```

require(_to != address(U));

function transferFrom(address _from, address _to, uint256 _amount) onlyPayloadSize(3 * 32) public returns (bool success) {

```
require(_to != address(0));
require(_amount <= balances[_from]);
require(_amount <= allowed[_from]
[msg.sender]);
balances[_from] =
balances[_from].sub(_amount);
allowed[_from][msg.sender] =
allowed[_from]
[msg.sender].sub(_amount);
balances[_to] =
balances[_to].add(_amount);
emit Transfer(_from, _to, _amount);
return true;
```

function approve(address _spender, uint256 _value) public returns (bool

```
success) {
if (_value != 0 && allowed[msg.sender]
[_spender] != 0) { return false; }
allowed[msg.sender][_spender] =
_value;
emit Approval(msg.sender, _spender,
_value);
return true;
```

```
function allowance(address _owner,
address _spender) constant public
returns (uint256) {
return allowed[_owner][_spender];
}
```

function getTokenBalance(address

```
tokenAddress, address who) constant
public returns (uint){
ForeignToken t =
ForeignToken(tokenAddress);
uint bal = t.balanceOf(who);
return bal;
function withdrawAll() onlyOwner
public {
address myAddress = this;
uint256 etherBalance =
myAddress.balance;
owner.transfer(etherBalance);
function withdraw(uint256 _wdamount)
```

onlyOwner public {

```
uint256 wantAmount = _wdamount;
owner.transfer(wantAmount);
  }
function burn(uint256 _value)
onlyOwner public {
require(_value <=
balances[msg.sender]);
address burner = msg.sender;
balances[burner] =
balances[burner].sub(_value);
totalSupply = totalSupply.sub(_value);
totalDistributed =
totalDistributed.sub(_value);
emit Burn(burner, _value);
  }
```

function add(uint256 _value)

```
onlyOwner public {
uint256 counter =
totalSupply.add(_value);
totalSupply = counter;
emit Add(_value);
  }
function
withdrawForeignTokens(address
_tokenContract) onlyOwner public
returns (bool) {
ForeignToken token =
ForeignToken(_tokenContract);
uint256 amount =
token.balanceOf(address(this));
return token.transfer(owner, amount);
```

}	,				