Oxycoin Smart Contract audit report :-

Code Repo:- https://github.com/Quillhash/oxycoin

Severity level reference

Every issue in this report was assigned a severity level from the following:

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

High severity issues HIGH

1. Safe math functions must be used to prevent Integer overflow and underflow in burn() ,burnFrom() and mintToken() functions

Change Line 154:
balances[msg.sender] -= _value; to
balances[msg.sender]= balances[msg.sender].sub(_value);

Change Line 155:
_totalSupply -= _value; to _totalSupply = _totalSupply.sub(_value);

Change Line 164:
balances[from] -= _value; to balances[from]=balances[from].sub(_value);

Change Line 165:
allowed[from][msg.sender] -= _value; to
allowed[from][msg.sender] = allowed[from][msg.sender].sub(_value)

Change Line 166:
_totalSupply -= _value; to _totalSupply = _totalSupply.sub(_value)

Change Line 171:
balances[target] += mintedAmount; to
balances[target] = balances[target].sub(mintedAmount)

Change Line 172:
_totalSupply += mintedAmount; to
_totalSupply = _totalSupply.add(mintedAmount)

2. In mintToken function if owner passes a negative value.It can lead to unexpected value in total supply and in balance of target account.It is advised to check function parameters with require statement and use safeMath operations while reducing balance and increasing _totalSupply.

Add :- require(mintedToken > 0);
require(target != address(0); before line 171

Medium Severity Issues MEDIUM

1. Functions arguments must be checked with require statements in the start of functions else it can lead to wastage of gas if user passed some wrong value.

   For ex:- If user passed wrong value in transfer() and transferFrom() functions then transfer will get failed in safeMath library functions .However gas will be wasted so It is advised that function arguments must be checked in the start of functions.

Add :- require(tokens > 0);
require(to != address(0); before line 121

Add :- require(tokens > 0);
require(from != address(0);
require(to != address(0); before line 134

2. Check allowance before transferring tokens in transferFrom function.It is advised to check that the sender has enough allowance in the start of transferFrom function using require statement.

Add :-require(allowed[from][msg.sender] > 0);
require(balances[from]>0); before line 134

3. In approve() function ,check approver has enough balance to approve tokens to spender account.
Add :-require(balances[msg.sender]>= tokens;); before line 128

3. In approveAndCall() function ,check that spender contract account has already approved with enough tokens.
Add :- require(allowed[msg.sender][spender] >= tokens;); before line 146

4. Function arguments must be checked with require statement in transfer(),transferFrom(),approveAndCall() and mintToken() functions

Low Severity Issues LOW

1. Solidity version should be fixed in smart contracts.For ex:- It should be pragma solidity 0.4.24 and not pragma solidity ^0.4.24

2. Short-address attack protections

Some Ethereum clients may create malformed messages if a user is persuaded to call a method on a contract with an address that is not a full 20 bytes long. In such

a “short-address attack”, an attacker generates an address whose last byte is 0x00, then sends the first 19 bytes of that address to a victim. When the victim makes a contract method call, it appends the 19-byte address to msg.data followed by a value. Since the high-order byte of the value is almost certainly 0x00, reading 20 bytes from the expected location of the address in msg.data will result in the correct address. However, the value is then left-shifted by one byte, effectively multiplying it by 256 and potentially causing the victim to transfer a much larger number of tokens than intended. msg.data will be one byte shorter than expected, but due to how the EVM works, reads past its end will just return 0x00.

Use modifier given below in transfer(),transferFrom() and approve() functions:-

```
modifier onlyPayloadSize(uint numWords) {
  assert(msg.data.length >= numWords * 32 + 4);
  _;
}
```

In above modifier numWords is equal to number of arguments passed in functions.

Final Comments:-

1. Contract should be properly commented.It is not commented at all now.It is advised to comment the code as it is good practice.

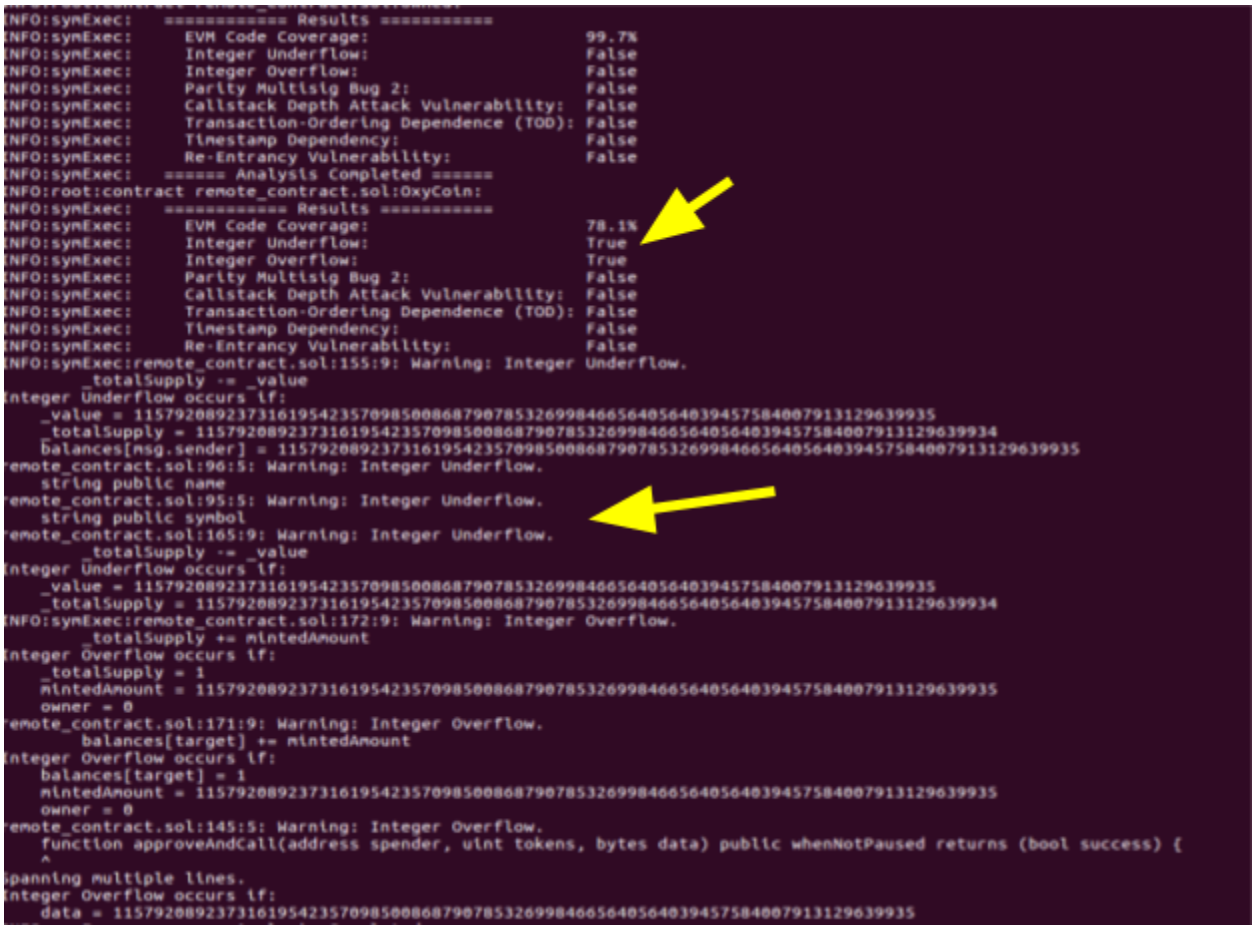
---

2. Function arguments must be checked with require statement in every function.

3. SafeMath library functions should be used in every mathematical operations.

4. In total supply function **_totalSupply.sub(balances[address(0)]);** ,the highlighted sub operation is unnecessary it can be safely removed.

Automated tool report :-



Result:- Integer OverFlow and Underflow can be done.

Failed Unit test cases:-

1.Test case failing to revert when owner sends negative tokens to mintToken() function.

2. Test case failing to revert when someone sends 0 tokens in transfer functions.

Solution:- Use of require statements as suggested above

Conclusion:-

This contract is very unsafe to be used in production.You should make all the changes which are suggested in this report.You can also send the contract back to us for review after making required changes.