SECURITY AUDIT

techracers

Link to code: https://etherscan.io/address/0x53148bb4551707edf51a1e8d7a93698d18931225#code

Code lines marked in PURPLE

Major

1. No ERC223 compatibility. Tokens get stuck in contracts UNKNOWINGLY when using ERC20.

2. A lot of failed transactions on etherscan because of the require break in transfer --> require(balancesCanSell[msg.sender])
- Users should be made aware of the require break, 'Can only transfer after 24th January', if they have a dapp or a newsletter. Unnecessary gas wastage for users.

3. Similarly, for approve + transferFrom, users will have lots of failed transactions due to the require break similar to the one in point 2.

4. SafeERC20 library imported but not used anywhere --> Unnecessary deployment gas cost.

5. In function
   function approveAndCall(address _spender, uint256 _value, bytes _extraData) returns (bool success)

   → The line

   require(_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))), msg.sender, _value, this, _extraData))

   Untrusted external calls to contracts, i.e _spender should be marked. I.e, _untrustedRecipient instead of _spender

Minor

1. Wrong Indentation almost everywhere. Makes the code difficult to read

2. freezeAccount(address target, bool canSell) is a misleading name should be --> toggleAccountFreeze(address target, bool canSell) - as owner can also unfreeze accounts using the function.

3. Similarly, event FrozenFunds(address target, bool frozen) is also a misleading name --> bool frozen can also be false

4. **Ignore** if not that time dependent - block numbers are more accurate.

   Should burn token be a public function ?
   No need for the function 'getBlockTimestamp'.

   Misleading naming convention `balancesCanSell`
   Indentations in code
   Improper commenting techniques

   Solidity version should be locked. pragma solidity ^0.4.13 (line1), following warning could be avoided- "sha3" has been deprecated in favour of "keccak256".

   No address(0) check for input parameter- target, in function freezeAccount(address target, bool canSell) (line 286)

   No address(0) check for input parameter- _spender, in function approveAndCall(address _spender, uint256 _value, bytes _extraData) (line 298)

   No check for _spender attribute to be a contract in function approveAndCall(address _spender, uint256 _value, bytes _extraData) (line 298)

Insted following function should be used:

```
function isContract(address addr) returns (bool) {
  uint size;
  assembly { size := extcodesize(addr) }
  return size > 0;
}
```

Line 302,
require(_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))), msg.sender, _value, this, _extraData));
Can be used to introduce re-entrancy, though it will not be a problem right now because value is assigned directly, but that itself is wrong. Also did not find any receiveApproval function in the provided code, therefore cannot check it.

_value has been directly assigned in allowed mapping, instead add function of safeMath library should be used. Why should we assign value directly, suppose a case where a user has allowed 1000 tokens, now he wants to add 500 more tokens to the value, upon calling approveAndCall function with 500, now the allowed mapping will be set to 500, instead of 1500 tokens.function allowed[msg.sender][_spender] = _value (line 299)

No check for msg.sender token balance before increasing allowed mapping. function allowed[msg.sender][_spender] = _value (line 299)

Too many failed transactions on etherscan.io due to the condition `require(now>dateDefrost);`. I guess people are not aware about the condition

Automated Analysis

1. **Oyente:**

```
INFO:root:Contract
/home/rails/work/audit/contracts/Peculium.sol:BasicToken:
INFO:symExec:Running, please wait...
INFO:symExec:    ============ Results ===========
INFO:symExec:      EVM code coverage:      99.8%
INFO:symExec:      Callstack bug:      False
INFO:symExec:      Money concurrency bug: False
INFO:symExec:      Time dependency bug:      False
INFO:symExec:      Reentrancy bug:      False
INFO:root:Contract
/home/rails/work/audit/contracts/Peculium.sol:BurnableToken:
INFO:symExec:Running, please wait...
INFO:symExec:    ============ Results ===========
INFO:symExec:      EVM code coverage:      99.9%
INFO:symExec:      Callstack bug:      False
INFO:symExec:      Money concurrency bug: False
INFO:symExec:      Time dependency bug:      False
INFO:symExec:      Reentrancy bug:      False
INFO:root:Contract
/home/rails/work/audit/contracts/Peculium.sol:Ownable:
INFO:symExec:Running, please wait...
INFO:symExec:    ============ Results ===========
INFO:symExec:      EVM code coverage:      99.5%
INFO:symExec:      Callstack bug:      False
INFO:symExec:      Money concurrency bug: False
INFO:symExec:      Time dependency bug:      False
INFO:symExec:      Reentrancy bug:      False
INFO:root:Contract/home/rails/work/audit/contracts/Peculium.sol:Saf
eERC20:
INFO:symExec:Running, please wait...
```

INFO:symExec: ========== **Results** ==========
INFO:symExec: EVM code coverage: 100.0%
INFO:symExec: Callstack bug: False
INFO:symExec: Money concurrency bug: False
INFO:symExec: Time dependency bug: False
INFO:symExec: Reentrancy bug: False
INFO:root:Contract/home/rails/work/audit/contracts/Peculium.sol:SafeMath:
INFO:symExec:Running, please wait...
INFO:symExec: ========== **Results** ==========
INFO:symExec: EVM code coverage: 100.0%
INFO:symExec: Callstack bug: False
INFO:symExec: Money concurrency bug: False
INFO:symExec: Time dependency bug: False
INFO:symExec: Reentrancy bug: False
INFO:root:Contract/home/rails/work/audit/contracts/Peculium.sol:StandardToken:
INFO:symExec:Running, please wait...
INFO:symExec: ========== **Results** ==========
INFO:symExec: EVM code coverage: 99.9%
INFO:symExec: Callstack bug: False
INFO:symExec: Money concurrency bug: False
INFO:symExec: Time dependency bug: False
INFO:symExec: Reentrancy bug: False
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======
INFO:symExec: ====== Analysis Completed ======

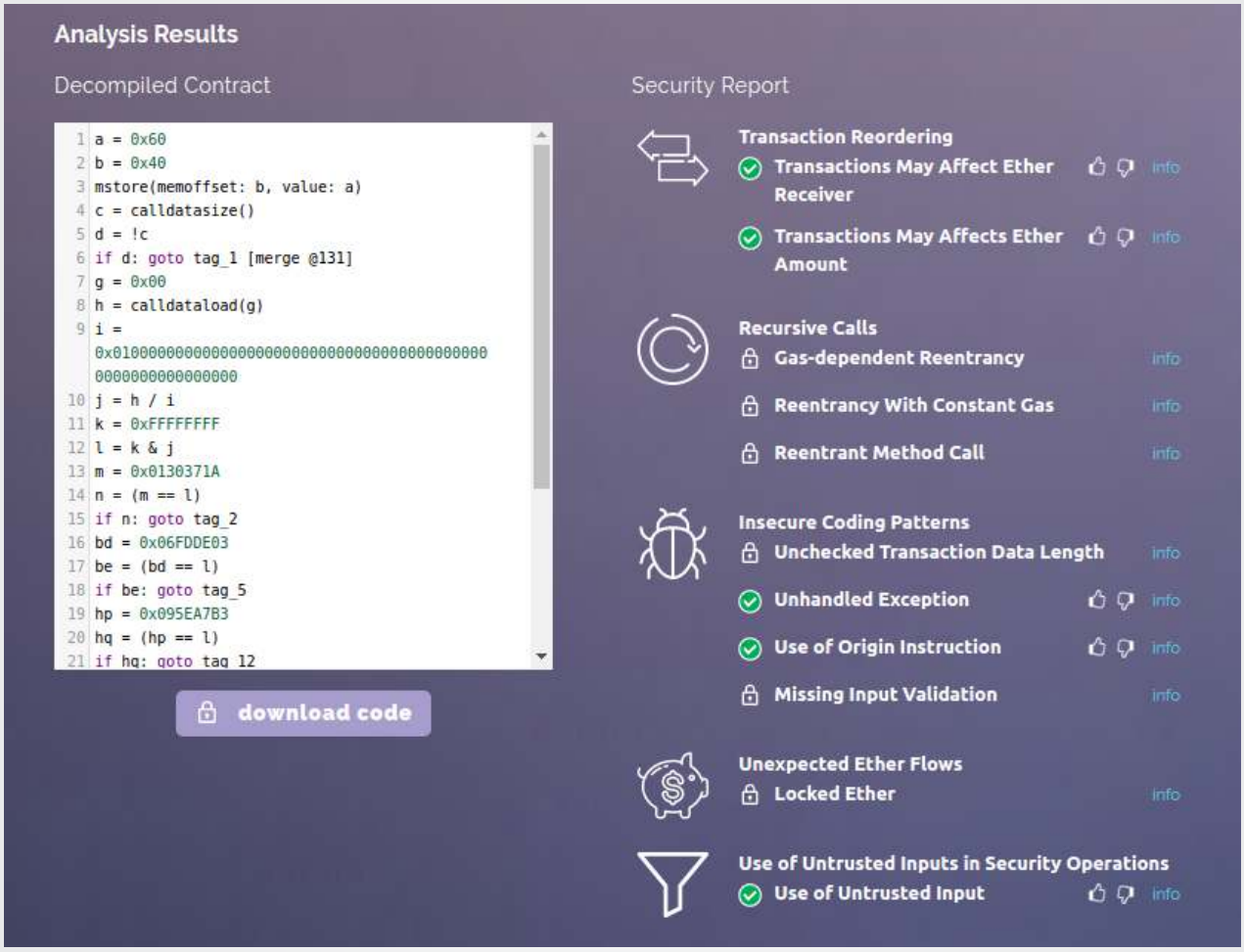
2. **Mythril:**

==== CALL with gas to dynamic address ====
Type: Warning
Contract: MAIN
Function name: approveAndCall(address,uint256,bytes)
PC address: 4784
The function approveAndCall(address,uint256,bytes) contains a function call to an address provided as a function argument. The available gas is forwarded to the called contract. Make sure that the logic of the calling contract is not adversely affected if the called contract misbehaves (e.g. reentrancy).

==== Integer Underflow ====
Type: Warning
Contract: MAIN
Function name: approveAndCall(address,uint256,bytes)
PC address: 4744
A possible integer underflow exists in the function approveAndCall(address,uint256,bytes).
The SUB instruction at address 4744 may result in a value < 0.

++++ Debugging info ++++
(32) - (calldata_MAIN_4 + calldata_MAIN_32 + 32 + 4 +
32 +
32 +
32 +
4 +
96 +
32 +
UDiv(31 + calldata_MAIN_4 + calldata_MAIN_32 + 32 + 4, 32)*
32).]

3. Securify:


Analysis Results
Decompiled Contract
1 a = 0x60
2 b = 0x40
3 mstore(memoffset: b, value: a)
4 c = calldatasize()
5 d = !c
6 if d: goto tag_1 [merge @131]
7 g = 0x00
8 h = calldataload(g)
9 i =
0x0100000000000000000000000000000000000000000000000000000000000000
10 j = h / i
11 k = 0xFFFFFFFF
12 l = k & j
13 m = 0x0130371A
14 n = (m == l)
15 if n: goto tag_2
16 bd = 0x06FDDE03
17 be = (bd == l)
18 if be: goto tag_5
19 hp = 0x095EA7B3
20 hq = (hp == l)
21 if hq: goto tag_12
download code
Security Report
Transaction Reordering
Transactions May Affect Ether Receiver
info
Transactions May Affects Ether Amount
info
Recursive Calls
Gas-dependent Reentrancy
info
Reentrancy With Constant Gas
info
Reentrant Method Call
info
Insecure Coding Patterns
Unchecked Transaction Data Length
info
Unhandled Exception
info
Use of Origin Instruction
info
Missing Input Validation
info
Unexpected Ether Flows
Locked Ether
info
Use of Untrusted Inputs in Security Operations
Use of Untrusted Input
info