QURAS TOKEN (XQC)
Audit Report

Conducted in Sep 2019

1. Executive Summary

QURAS is a secret contract platform that fulfills privacy protection needs. Before its release, QURAS team decided to issue ERC20 token named “QURAS TOKEN”. TECHFUND, a global technology accelerator which supported several hundreds of startups and large companies’ new businesses in various domains worldwide mainly from technology aspect, had also conducted security audits and was asked to specify the token’s issues from QURAS team.
There were 1 low level issue, 4 note level issues and 2 non-risk findings (just for confirmation). After the report, we TECHFUND reaudited the revision and verified all the issues above were solved.

2. Scope

i . Compiler Version

The target token contract first used solidity version 0.4.18 and then it's updated into 0.5.11.
So we checked the two versions if there exist any known issues dependent on compiler version in the source code.

ii . Source Code

The target token's source code is provided by QURAS team as below (it's a private repository as of 30th Sep) on 13th Sep, 2019.
https://github.com/DonJ92/XQC
The token is ERC20-Compliant as explained in a following section "Function Structure".
And the contract is composed of a single file and doesn't use any other external files.
Any framework such as OpenZeppelin isn't utilized.

iii . Deployment Setting

We verified the token's deployment settings, such as Gas Price, Gas Limit, to ensure the token is properly deployed without any loss.

3. Source Summary

i . Function Structure

+ SafeMath
    - safeAdd
    - safeSub
    - safeMul
    - safeDiv

+ ERC20Interface
    - totalSupply
    - balanceOf
    - allowance
    - transfer*
    - approve*
    - transferFrom*

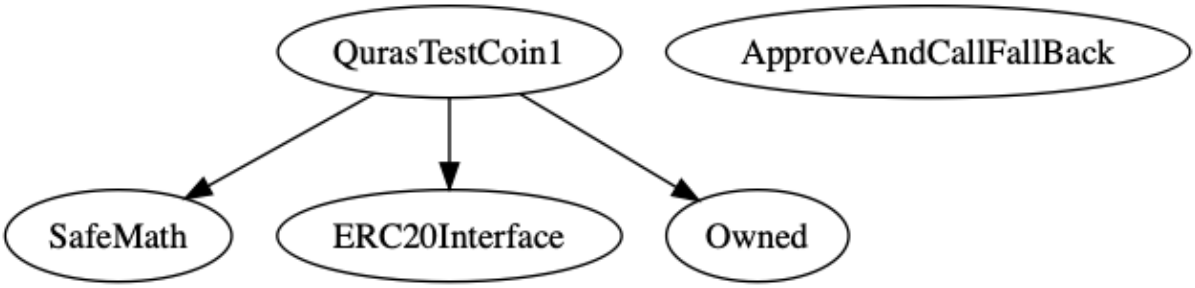
+ ApproveAndCallFallBack
    - receiveApproval*

+ Owned
    - <Constructor>*
    - transferOwnership*
    - acceptOwnership*

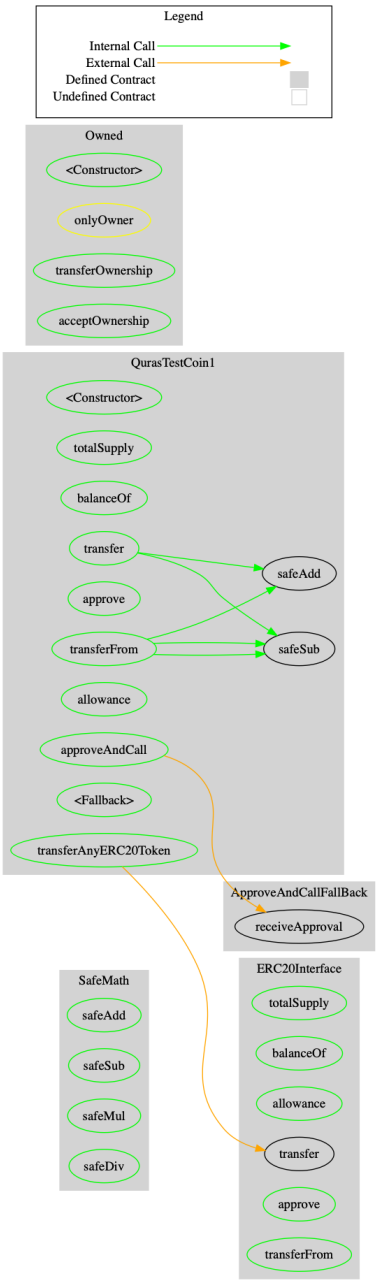
+ QurasCoin (ERC20Interface, Owned, SafeMath)
    - <Constructor>*
    - totalSupply
    - balanceOf
    - transfer*
    - approve*
    - transferFrom*
    - allowance
    - approveAndCall*
    - <Fallback>@
    - transferAnyERC20Token*

@ = payable function
* = non-constant function

ii . Inheritance Graph



iii . Call Graph



4. Automated security analysis

We described automated security analysis result for each contract and no vulnerability was found.

- Owned

EVM Code Coverage:	99.7%
Integer Underflow:	False
Integer Overflow:	False
Parity Multisig Bug 2:	False
Call stack Depth Attack Vulnerability:	False
Transaction-Ordering Dependence (TOD):	False
Timestamp Dependency:	False
Re-Entrancy Vulnerability:	False

- QurasCoin

EVM Code Coverage:	72.5%
Integer Underflow:	False
Integer Overflow:	False
Parity Multisig Bug 2:	False
Call stack Depth Attack Vulnerability:	False
Transaction-Ordering Dependence (TOD):	False
Timestamp Dependency:	False
Re-Entrancy Vulnerability:	False

- SafeMath

EVM Code Coverage:	98.3%
Integer Underflow:	False
Integer Overflow:	False
Parity Multisig Bug 2:	False
Call stack Depth Attack Vulnerability:	False
Transaction-Ordering Dependence (TOD):	False
Timestamp Dependency:	False
Re-Entrancy Vulnerability:	False

5. Detailed Analysis and Suggestions

These are the results of detailed analysis and suggestions derived from it. There are totally 7 findings, which were later fixed and confirmed to have no security issue.

Risk Level: Critical	0
Risk Level: High	0
Risk Level: Middle	0
Risk Level: Low	1
Risk Level: Note	4
Risk Level: None	2

5.1. Fix a compiler version to use

Risk Level: None

pragma solidity ^0.4.18; // compiles with 0.4.18 and above
pragma solidity 0.4.18; // compiles with 0.4.18 only

```
1   pragma solidity ^0.4.18;
2
3
```

It is suggested to use particular version as there might be changes in future versions of a compiler which cannot be checked for now and might lead to breaking changes in deployed smart contract.

5.2. Constant is deprecated in 0.4.17

Risk Level: Note

In compiler version 0.4.17 and above, constant has been deprecated and view is suggested in place of it.
Constant indicates that network verification won't be necessary. Callers receive return values instead of transaction hashes. Starting with solc 0.4.17, constant is deprecated in favor of two new and more specific modifiers.
As a rule of thumb, use view if your function does not modify storage and pure if it does not even read any state information.

[https://github.com/ethereum/solidity/releases/tag/v0.4.17]

Version 0.4.17

chriseth released this on Sep 21, 2017 · 5625 commits to release since this release

As we are getting closer to the next breaking release, we want to give everyone a heads up by introducing `pragma experimental "v0.5.0"` which already enables some of the new safety features of the 0.5.0 release.

Furthermore, this release finally checks the modifiers `view` (used to be named `constant`) and `pure` on functions. As a rule of thumb, use `view` if your function does not modify storage and `pure` if it does not even read any state information - but the compiler will also suggest the tightest restriction itself.

We also worked further on the new ABI encoder: Functions can now return structs. Switch it on using `pragma experimental ABIEncoderV2`. It should already work, but still generates more expensive code.

Finally, many new warnings were introduced and error messages improved.

5.3. Fallback function is not required

Risk Level: Note

In compiler version 0.4.0 and above, contracts automatically revert payments, making fallback function redundant. There is no need to add that in the code.

```
function () public payable {
    revert();
}
```

5.4. ERC20 Approve vulnerability

Risk Level: Low

Someone already approved can acquire extra allowance when a token holder changes the allowance for him/her.

Here is possible attack scenario:

1. Alice allows Bob to transfer N of Alice's tokens (N>0) by calling approve method on Token smart contract passing Bob's address and N as method arguments
2. After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls approve method again, this time passing Bob's address and M as method arguments

3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls transferFrom method to transfer N Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another M tokens
5. Before Alice noticed that something went wrong, Bob calls transferFrom method again, this time to transfer M Alice's tokens.

So, Alice's attempt to change Bob's allowance from N to M (N>0 and M>0) made it possible for Bob to transfer N+M of Alice's tokens, while Alice never wanted to allow so many of her tokens to be transferred by Bob.

If there is possibility to use approve function for not fully trusted party, it's recommend to replace the approve function with following functions. They atomically check allowance and approve. This process is often included in tokens recently (from 2018) issued, though old tokens (before 2017) don't have such process.

```
  function increaseAllowance(address spender, uint256 addedValue) public
returns (bool) {
      _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].add(addedValue));
      return true;
  }
  function decreaseAllowance(address spender, uint256 subtractedValue) public
returns (bool) {
      _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].sub(subtractedValue, “ERC20: decreased
allowance below zero”));
      return true;
  }
  function approve(address spender, uint256 amount) public returns (bool) {
      _approve(_msgSender(), spender, amount);
      return true;
  }
  function _approve(address owner, address spender, uint256 amount) internal {
      require(owner != address(0), “ERC20: approve from the zero address”);
      require(spender != address(0), “ERC20: approve to the zero address”);
      _allowances[owner][spender] = amount;
      emit Approval(owner, spender, amount);
  }
```

5.5. Insufficient Gas Price

Risk Level: Note

Gas Price is not high enough to smoothly deploy the contract and if you redeploy it, there might be two duplicated tokens.
You seem to refer to Ropsten Gas Price, but Mainnet one is different. So it's recommended to check below site for Gas Price and properly set.
https://ethgasstation.info

5.6. Uncommon decimal

Risk Level: Note

The value of "decimals" is currently 8, though 18 is the most common and some wallets don't accept non-18 decimals tokens.
For example, this wallet hadn't accepted such non standard token until July 24, 2019.
https://appgrooves.com/app/liquidity-network-wallet-by-liquidchain-gmbh

5.7. Possibility of mismatch between the token name and symbol

Risk Level: None

The token name seems to have changed from "Quras Coin" to "Quras Token". However, the symbol name hasn't changed from "XQC". If "C" in "XQC" means "Coin", it might be better to rename "XQT".