SECURITY AUDIT OF QUARKCHAIN TOKEN SMART CONTRACTS

REPORT

MAY 30, 2018

✔erichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology >> Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on May 30, 2018. We would like to thank QuarkChain Foundation to trust Verichains Lab to audit smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the smart contracts. The scope of the audit is limited to the source code files provided to Verichains Lab on May 28, 2018. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

The assessment identified few issues in QuarkChain smart contracts code. Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

CONTENTS

ACRONYMS AND ABBREVIATIONS

Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
ETH (Ether)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
EVM	Ethereum Virtual Machine.

AUDIT OVERVIEW

ABOUT QUARKCHAIN

QuarkChain is an innovative permission-less blockchain architecture that aims to meet the global-wise commercial standard. It provides a secure, decentralized, and scalable blockchain solution to deliver 100,000+ on-chain TPS. The main features of QuarkChain are: reshardable two-layered blockchain, guaranteed security by market-driven collaborative mining, anti-centralized horizontal scalability, efficient cross-shard transactions, simple account management and turing-complete smart contract platform.

The website of QuarkChain is at https://quarkchain.io/
White paper (EN – version 0.3.4) is at https://quarkchain.io/QUARK CHAIN Public Version 0.3.4.pdf

About QuarkChain Token

Token Name	QuarkChain Token
Symbol	QKC
Decimals	18
Total Tokens	10 Billion QKC
Token Distribution	
Team	1.5 Billion QKC (15% - lockup to 2 years)
Foundation	1.5 Billion QKC (15% - lockup to 2 years)
Advisor	500 Million QKC (5% - lockup to 2 years)
Mining, Community & Marketing	4.5 Billion QKC (45% - lockup to 2 years)
Token Sale	2 Billion QKC • Private sale: 1.6 Billion QKC (16% - Hard cap: USD $16M) • Public sale: 400 Million QKC (4% - Hard cap: USD $4M)
Individual Cap	Progressive Individual Cap. After the first 12 hours of the crowd sale, whitelisted investor personal cap will double every 6 hours, until the tokens are depleted or the crowd sale ends.
Gas Limit	50 Gwei

SCOPE OF THE AUDIT

This audit focused on identifying security flaws in code and the design of the smart contracts for the upcoming Crowdsale and Token.

It was conducted on commit *`0121dbb451804ff393fc264340dc5e80da9bc8eb`* of branch *`master`* from GitHub repository of QuarkChain Token.

Repository URL:
https://github.com/QuarkChain/quarkchain-token/tree/0121dbb451804ff393fc264340dc5e80da9bc8eb

The scope of the audit is limited to the following 5 source code files:

Source File	**SHA256 Hash**
QuarkChainToken.sol	2e9f743e0f16d5561b5209950f6da755b15fee5cb916108c80c6cffb3f4b30a7
QuarkChainCrowdsale.sol	d75737323b930788e71bf92ee5873906d3133a2544a2cf490ee7c5c61665fefe
ProgressiveIndividualCappedCrowdsale.sol	1eea840c797b0ff7f7e3fcf7919ae9eb58d02341a5079664b2e13065341e1fac
Migrations.sol	5c36fd60f54bbabcb3eca9e1cc5c21c2668f2181fa4658a22d5c865c227e3d81

QuarkChain's crowdsale and token contracts are based mainly on OpenZeppelin, the industry-standard library with extra features including IndividuallyCappedCrowdsale, CappedCrowdsale, TimedCrowdsale, Pausable, SafeMath.

Note that initialized parameters for crowdsale contracts including start and end time, hard cap, personal cap... will be passed to the contract when these contracts are created so verifying the correctness of initialized parameters is not possible at the time of this audit.

AUDIT METHODOLOGY

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- TimeStamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Dos with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories, depending on their criticality:

LOW	An issue that does not have a significant impact, can be considered as less important
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.

AUDIT RESULT

VULNERABILITIES FINDINGS

MEDIUM
NON-REQUIRED INTEGER OVERFLOW BLOCKING

In ProgressiveIndividualCappedCrowdsale, function getCurrentEthCapPerAddress is called by _preValidatePurchase which is called before any purchase:

```
/**
    * @dev Calculate personal cap based on time. First 12hr it's 1x, then double the cap
    * ever 6 hours after.
    * @return Calculated personal cap based on time
*/
function getCurrentEthCapPerAddress() public view returns (uint256) {
    if (now < startTime) {
        return 0;
    }

    uint256 timeSinceStart = now.sub(startTime);
    if (timeSinceStart < FIRST_PERIOD) {
        // Only allow the base personal cap.
        return caps[msg.sender];
    }

    uint256 currentPeriod = (timeSinceStart - FIRST_PERIOD).div(PERIOD_INTERVAL).add(1);
    // In case of being called way after the token sale, which shouldn't happen.
    if (currentPeriod >= 256) {
        return 0;
    }
    return caps[msg.sender].mul(2 ** currentPeriod);
}
```

The returned value is calculated by mul function of SafeMath, which reverts if the operation overflowed. In this case, the cap calculation will always revert if the crowdsale is running long-enough. If the origin caps is **1 ETH** (10^{18} wei), maximum possible periods is about **192** before the multiplication is overflowed, which is equals to **48 days** - that is quite practical (no one could buy Tokens after 48 days due to the overflow).

Test case output (refer to Appendix II - Personal Cap Overflow testcase):

```
  Contract: token sale
    ✓ should not overflow with short periods (1078ms)
    1) should not overflow with long periods

    Events emitted during test:
    ---------------------------

    Transfer(from: <indexed>, to: <indexed>, value: 1.6e+27)
    Transfer(from: <indexed>, to: <indexed>, value: 5e+26)
    Transfer(from: <indexed>, to: <indexed>, value: 1.5e+27)
    Transfer(from: <indexed>, to: <indexed>, value: 6e+27)

    ---------------------------


  1 passing (4s)
  1 failing

  1) Contract: token sale
       should not overflow with long periods:
     AssertionError: expected promise not to be rejected with an error including 'VM
Exception while processing transaction: revert'
```

Recommended Fixes

- Return the maximum unsigned integer value in case of multiplication overflow, for example

```
if(caps[msg.sender]==0) return 0;
// if the multiplication will overflow, return MAX_UINT256
if(caps[msg.sender] >= 2**(256 - currentPeriod)) return ~uint256(0);
return caps[msg.sender] * 2**currentPeriod;
```

LOW
LACK OF INPUT VALIDATION FOR SETCROWDSALEADDRESS

```
function setCrowdsaleAddress(address _crowdsaleAddress) external onlyOwner {
    // Can only set one time.
    if (crowdsaleAddress == 0x0) {
        crowdsaleAddress = _crowdsaleAddress;
        balances[crowdsaleAddress] = INITIAL_SUPPLY;
    }
}
```

The setCrowdsaleAddress function does not validate _crowdsaleAddress which can lead to (unintended) setting balances of 0x0 address to INITIAL_SUPPLY.

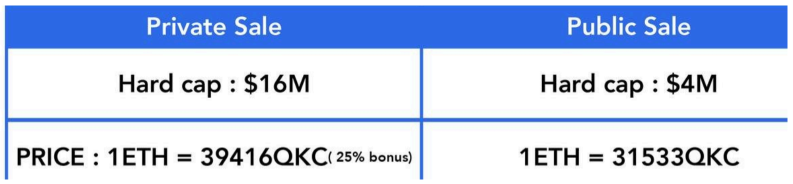
After calling setCrowdsaleAddress with address 0x0 parameter, the Owner could call setCrowdsaleAddress once again with a correct crowdsaleAddress to set the balance of that address to INITIAL_SUPPLY. In this case, the contract would have totally 2xINITIAL_SUPPLY Tokens = 20 Billion QRK. While this has no impact to the Crowdsale flows, this might lead to unintended and/or unexpected behavior in some applications using Tokens balance data in the future.

Recommended Fixes

- Add require(_crowdsaleAddress != 0x0); to ensure that balances of 0x0 wont be set.

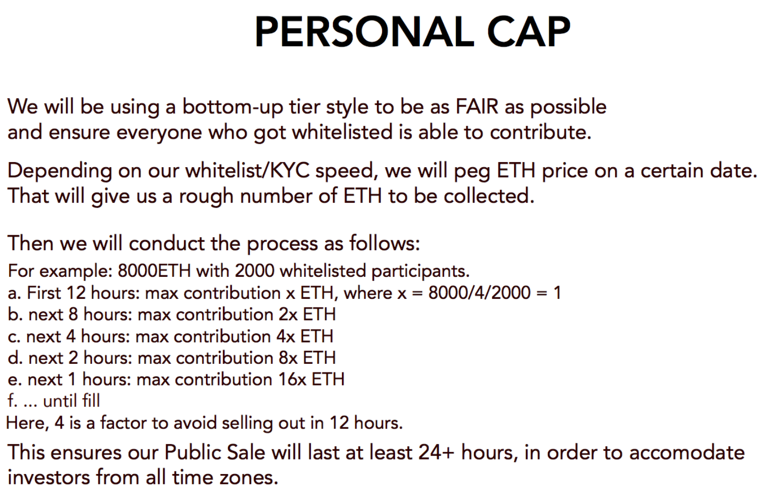
OTHER RECOMMENDATIONS / SUGGESTIONS

- Information about the token sale on the website at https://quarkchain.io/QuarkChain-Token-Sale-Terms[EN-ZH].pdf and blog https://steemit.com/quarkchain/@quarkchain/quarkchain-token-sale-terms-and-conditions are out of date with some inconsistencies. Recommend to update these documents to align with the token sale logic in the contract.
    - ETH price is very out of date (1 ETH around US$315)



Private Sale	Public Sale
Hard cap : $16M	Hard cap : $4M
PRICE : 1ETH = 39416QKC(25% bonus)	1ETH = 31533QKC

- Progressive Individual Cap calculation in the smart contract is quite different from what describes on the Token sale document and blog post.


PERSONAL CAP

We will be using a bottom-up tier style to be as FAIR as possible
and ensure everyone who got whitelisted is able to contribute.

Depending on our whitelist/KYC speed, we will peg ETH price on a certain date.
That will give us a rough number of ETH to be collected.

Then we will conduct the process as follows:
For example: 8000ETH with 2000 whitelisted participants.
a. First 12 hours: max contribution x ETH, where x = 8000/4/2000 = 1
b. next 8 hours: max contribution 2x ETH
c. next 4 hours: max contribution 4x ETH
d. next 2 hours: max contribution 8x ETH
e. next 1 hours: max contribution 16x ETH
f. ... until fill
Here, 4 is a factor to avoid selling out in 12 hours.
This ensures our Public Sale will last at least 24+ hours, in order to accomodate
investors from all time zones.

- The current code is already minimalized, consider adding documents for all modifiers and functions to ensure its correct usage: modifier onlyWhenTransferEnabled, validDestination of QuarkChainToken.

CONCLUSION

QuarkChain crowdsale and token smart contracts have been audited by Verichains Lab using various public and in-house analysis tools and intensively manual code review. The assessment identified some issues in QuarkChain smart contracts code. Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

LIMITATIONS

Note that initialized parameters for crowdsale contracts including start and end time, hard cap, personal cap... will be passed to the contract when these contracts are created so verifying the correctness of initialized parameters is not possible at the time of this audit.

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

APPENDIX I - CALL FLOWS

Figure 1 Call graph of QuarkChainToken.sol

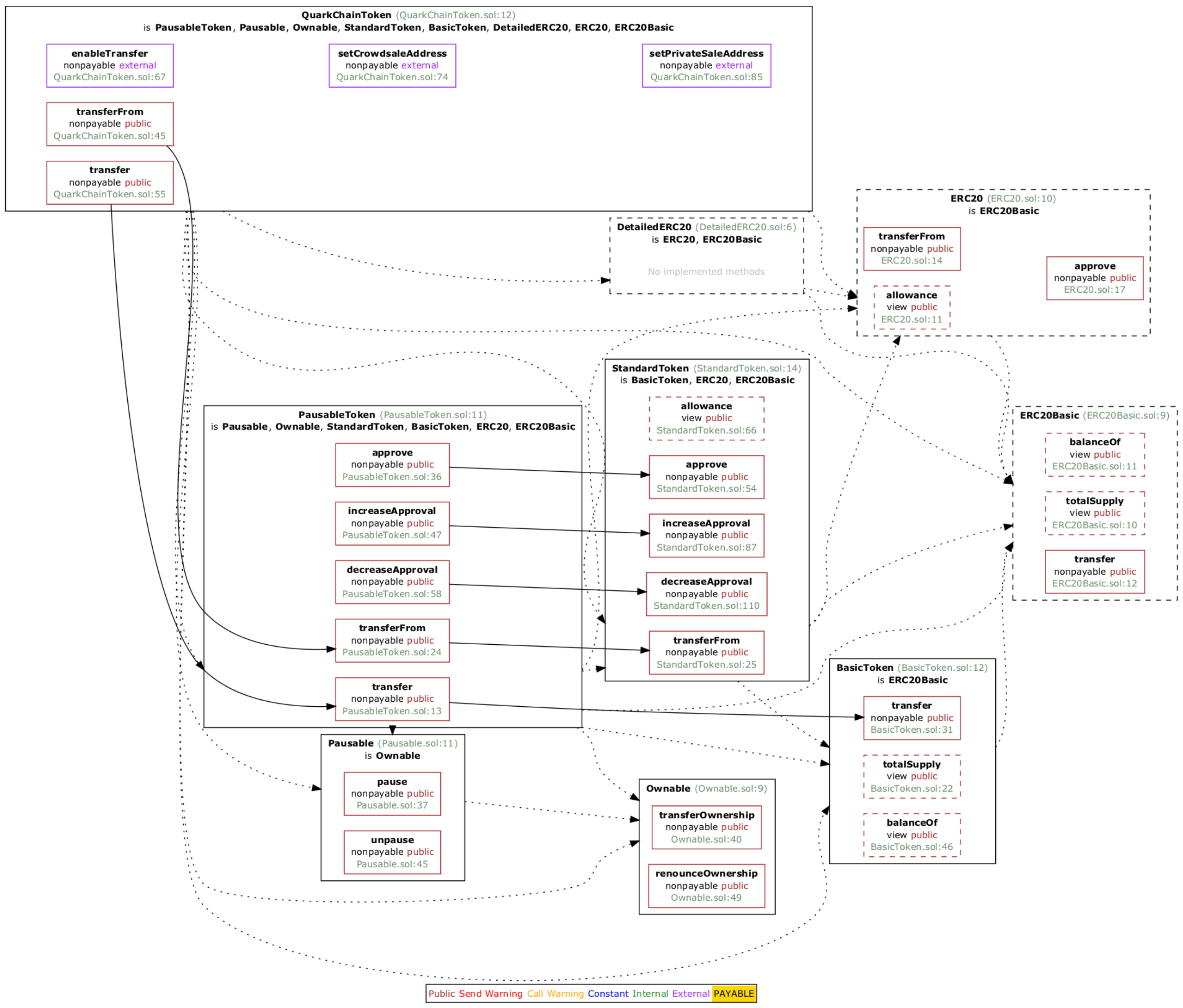
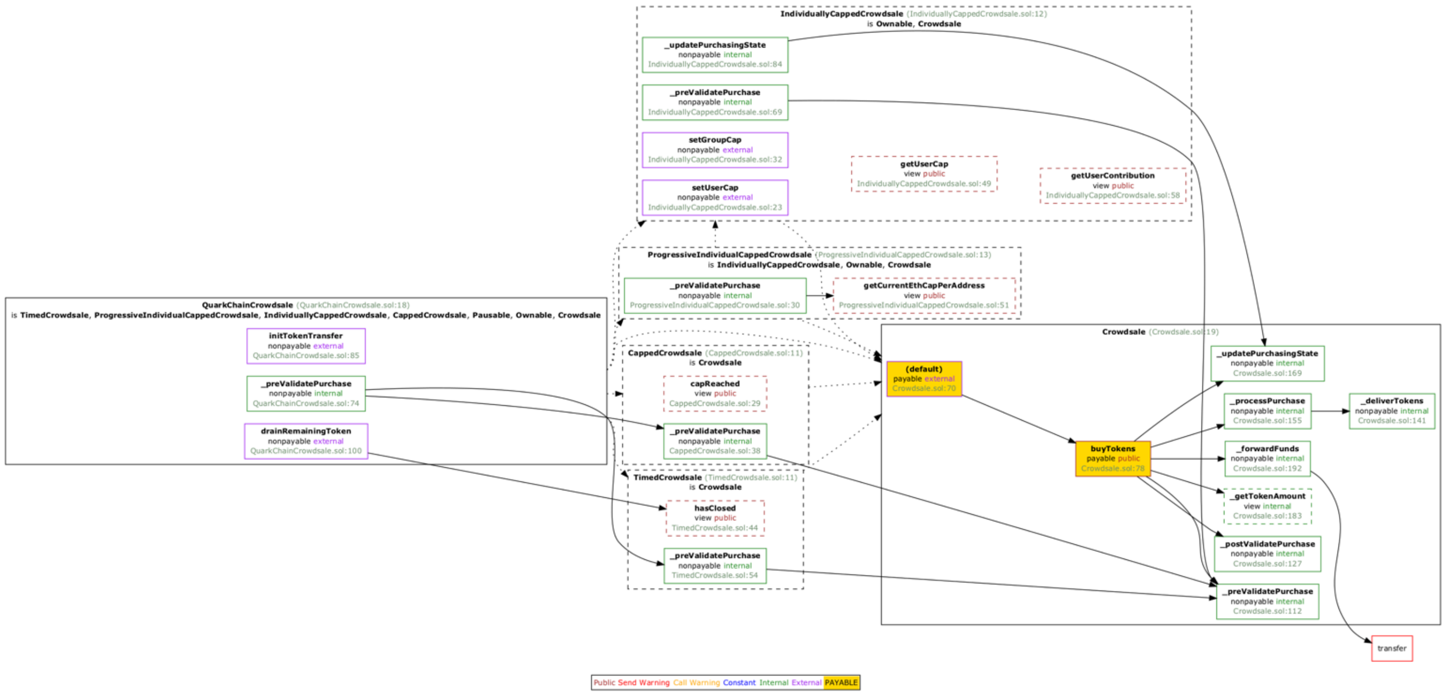

QuarkChainToken (QuarkChainToken.sol:12)
is PausableToken, Pausable, Ownable, StandardToken, BasicToken, DetailedERC20, ERC20, ERC20Basic
enableTransfer nonpayable external QuarkChainToken.sol:67
setCrowdsaleAddress nonpayable external QuarkChainToken.sol:74
setPrivateSaleAddress nonpayable external QuarkChainToken.sol:85
transferFrom nonpayable public QuarkChainToken.sol:45
transfer nonpayable public QuarkChainToken.sol:55
DetailedERC20 (DetailedERC20.sol:6) is ERC20, ERC20Basic
No implemented methods
ERC20 (ERC20.sol:10) is ERC20Basic
transferFrom nonpayable public ERC20.sol:14
approve nonpayable public ERC20.sol:17
allowance view public ERC20.sol:11
StandardToken (StandardToken.sol:14) is BasicToken, ERC20, ERC20Basic
allowance view public StandardToken.sol:66
approve nonpayable public StandardToken.sol:54
increaseApproval nonpayable public StandardToken.sol:87
decreaseApproval nonpayable public StandardToken.sol:110
transferFrom nonpayable public StandardToken.sol:25
PausableToken (PausableToken.sol:11) is Pausable, Ownable, StandardToken, BasicToken, ERC20, ERC20Basic
approve nonpayable public PausableToken.sol:36
increaseApproval nonpayable public PausableToken.sol:47
decreaseApproval nonpayable public PausableToken.sol:58
transferFrom nonpayable public PausableToken.sol:24
transfer nonpayable public PausableToken.sol:13
ERC20Basic (ERC20Basic.sol:9)
balanceOf view public ERC20Basic.sol:11
totalSupply view public ERC20Basic.sol:10
transfer nonpayable public ERC20Basic.sol:12
BasicToken (BasicToken.sol:12) is ERC20Basic
transfer nonpayable public BasicToken.sol:31
totalSupply view public BasicToken.sol:22
balanceOf view public BasicToken.sol:46
Pausable (Pausable.sol:11) is Ownable
pause nonpayable public Pausable.sol:37
unpause nonpayable public Pausable.sol:45
Ownable (Ownable.sol:9)
transferOwnership nonpayable public Ownable.sol:40
renounceOwnership nonpayable public Ownable.sol:49
Public Send Warning Call Warning Constant Internal External PAYABLE

Figure 2 Call graph of QuarkChainCrowdsale.sol



APPENDIX II - PERSONAL CAP OVERFLOW TESTCASE

```
// Tests inspired from Request.network :)
const assert = require('assert');
const BigNumber = require('bignumber.js');
require('chai').use(require('chai-as-promised')).should();

const Crowdsale = artifacts.require('./QuarkChainCrowdsale.sol');
const Token = artifacts.require('./QuarkChainToken.sol');

// Utility constants.
const day = 3600 * 24;
const gasPriceMax = 50000000000;
const revertError = 'VM Exception while processing transaction: revert';
const foundation = '0x8E2491C895f543b810e1675A1f262D860696889a';

// Utility helper functions.
async function addHoursOnEVM(hours) {
    const seconds = hours * 3600;
    await web3.currentProvider.send({
        jsonrpc: '2.0',
        method: 'evm_increaseTime',
        params: [seconds],
        id: 0,
    });
    await web3.currentProvider.send({
        jsonrpc: '2.0',
        method: 'evm_mine',
        params: [],
        id: 0,
    });
}

function txGen(from, value) {
    return {
        from,
        value,
        gasPrice: gasPriceMax,
    };
}

contract('token sale', (accounts) => {
    const investor1 = accounts[5];
    const investor2 = accounts[6];
    let crowdsale;
    let token;

    beforeEach(async() => {
        const currTs = web3.eth.getBlock(web3.eth.blockNumber).timestamp;
        const cap = web3.toWei(10);

        token = await Token.new();
        crowdsale = await Crowdsale.new(currTs + day, currTs + (day * 60), cap, token.address);
        // Required set-up.
        await token.setCrowdsaleAddress(crowdsale.address);
        assert(
            (await token.totalSupply()).equals(await token.balanceOf(crowdsale.address)),
            'wrong balance for token sale address after set up',
        );
        await crowdsale.initTokenTransfer();
        assert.equal(await crowdsale.initTransferred(), true, 'should have transferred tokens');
    });

    it('should not overflow with short periods', async() => {
        // Crowd sale starts.
        await addHoursOnEVM(25);

        await crowdsale.setUserCap(investor1, web3.toWei(1));

        // Skip 120 periods
        await addHoursOnEVM(6 * 120);
        await crowdsale.sendTransaction(txGen(investor1, web3.toWei(2)))
            .should.not.be.rejectedWith(revertError);
    });

    it('should not overflow with long periods', async() => {
        // Crowd sale starts.
        await addHoursOnEVM(25);

        await crowdsale.setUserCap(investor2, web3.toWei(1));

        // Skip 240 periods
        await addHoursOnEVM(6 * 240);
        await crowdsale.sendTransaction(txGen(investor2, web3.toWei(2)))
            .should.not.be.rejectedWith(revertError);
    });
});
```