SECURITY AUDIT OF CHAINTEX SMART CONTRACTS

PUBLIC REPORT

APR 17, 2019

✓erichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology >> Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on April 12, 2019. We would like to thank ChainTEX 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 April 05, 2019. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

The assessment identified some issues in ChainTEX 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 CHAINTEX SMARTCONTRACTS

ChainTex SmartContracts are an on-chain liquidity protocol allows instant exchange tokens on Tomochain network with similar function as KyberNetwork on Ethereum.

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 *8d5b849c0a84ad437420a8e7bd194d941d3c5fc9* of branch ***audit_v1*** from GitHub repository of ChainTEX Smart Contracts.

Repository URL:
https://github.com/chaintex/SmartContracts/tree/8d5b849c0a84ad437420a8e7bd194d941d3c5fc9

Source File	SHA256 Hash
Network.sol	4e72743254eefb0fdc121b54b599c4c6cacdd43ca9b0cd41f80eb58ec3dfe277
ConversionRates.sol	e6e7ffc732633b61e78bcae3a150b05657e618f36bd52e198a9fb1db4bcfc750
ExpectedRate.sol	9b5296c8572d2147fc814d4bda421ce9fb6545fbf4e0f46e3ab053ab57e2da57
FeeSharing.sol	cf3b8a32f4498606a3eba5c7925fc2e4f95a2ffe98622fcd3f88fc4d69827a7f
LiquidityConversionRates.sol	9b6d73887caa056956bf0624bc922319a1e019300d470b7cffe316b5cf8c6181
LiquidityFormula.sol	ea87ab96516392e8aa43d5b0dd644983b8d857331771e9872b89e80e965e0b16
NetworkProxy.sol	8b66fb69c31490ab2bf166534da0716eda935dc14f758df3ab0d499385cad58c
PermissionGroups.sol	3035157f8b9da69c8e6087d6cccc6aeb0b19847683c53ba3ba1591468066a6c8
Reserve.sol	a07240963aecc75720f6839e9fe1d52db96e1317c1a99b20b899c8237edfa7cb
SanityRates.sol	7cc00ae5a15394ee5ce85954ede9e89d2bcb1cc04332f1ecb7c26ab03e1fb24a
Utils.sol	2298949c6a1d2377b4f416388f0dcbbc7ba46a87ffd7ee81cc5677ee2c529fe9
Utils2.sol	45e7b81a11c097035d3bab21bdbd6a34a358166cf361641526a04d0adf68bd84
VolumeImbalanceRecorder.sol	8b78ed53fb71bcd93dcb01f3f48654dff70d635ef6a22b93bd7039f76daa5c48
WhiteList.sol	431c09cf027657cf0a7f23c5a375c275692c85bcd781be0661945573cb96e3db
Withdrawable.sol	1c9a570a96e39c671313b449d688e36b09dfcc475aaad93ec9dd279c9cafeee0
ConversionRatesInterface.sol	78990fa9ff7afddc285fcef7b2fd8d8f99ac66517b43c2c13d76ddb0f2a0c296
ExpectedRateInterface.sol	a51dc626e83fdb539496c9595b9cad11c461129fcf76755e2ecdd7d06295e6a5
FeeSharingInterface.sol	a7b4ddba30dd1b3a0b429f2c13e44d76d90421a8cdcfcce2e05b4e2910bfd73d
NetworkInterface.sol	4a3a910056286fec49c5e28aef03679ba8e0ede76986a4730356f98a92042a47
NetworkProxyInterface.sol	3219208bc7ef5637a203e544764c4cf2a8f8c7c4974b2d869d0fee60ed4c7f17
ReserveInterface.sol	0970e8c2ba1a8896cf796f1007321b465a1197962e94fc76b9766d248534b7f5
SanityRatesInterface.sol	57b2d9968a4eb0591d501f9511a524dff4c868669c844e2c85716b46ea5cff27
SimpleNetworkInterface.sol	0194b6ebdcbbbc2ecaabba0f476ce85cc214d34be50b718fbe16f58605f4cf85
TRC20Interface.sol	74bde3bd587f73bebcdffb317ef8438cf12fc4a477065962688e85532da66942
WhiteListInterface.sol	f8c89ac137f23d12d56883db197dd751b3b33216fe1a36963ff4f12004590d2a

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 RK87, our in-house smart contract security analysis tool.
- 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 RESULTS

ChainTEX Smart Contracts was forked from KyberNetwork smart contracts at https://github.com/KyberNetwork/smart-contracts/tree/f39b53df5929c558e789cf4fed71ed852f97ed10.

Most of the contracts in current audit version are quite similar to commit `1401bfb16891c7483b99305f8fbcfb0d3ebdef8b` from **KyberNetwork** repository, while **ExpectedRate** and **KyberNetwork** (renamed to **Network**) are most similar to commit `780d7a50e87c2f3303acb75e1ee7b0c314a29b22`.

Changes from **KyberNetwork** contracts to current ChainTEX contracts are summaried as below:

- Update solidity version from **0.4.18** to **0.4.25**:
  - Changed named constructors to **constructor**.
  - Added **memory** modifiers.
  - Changed **public** modifier to **external** modifier in interfaces.
  - Added **emit** to publish events.
- Renamed ERC20 contract to TRC20.
- Changed text/variable name from ETH to TOMO.
- Removed Kyber from text/variable name/contract name.
- In *ConversionRates.sol*:
  - Added function **getListedTokensAtIndex(uint id)**.
- In ExpectedRate.sol:
  - Added function **getExpectedFeeRate(TRC20 token, uint srcQty)**.
  - Added function **setNetwork** to allow changing the network.
- Added *FeeSharing.sol* and *FeeSharingInterface.sol*.
- In *Network.sol* (renamed from *KyberNetwork.sol*):
  - Added functions: **`payTxFee`**, **`addFeeReserve`**, **`setFeeSharing`**, **`setFeePercent`**, **`getExpectedFeeRate`**, **`getReservesPerTokenSrcCount`**, **`getReservesPerTokenDestCount`**, **`findBestRates`**, **`tradeFee`**, **`calcActualFeeAmounts`**.
  - Removed Kyber's fee related stuffs:
    - Function **`setFeeBurner`**.
  - In function **`getUserCapInWei`**:
    - Returns infinity (2^{255}) if **`whileListContract`** is 0x0.
  - In function trade:
    - Removed **`feeBurner`** calls:

```
        if (tradeInput.src != TOMO_TOKEN_ADDRESS)
            require(feeBurnerContract.handleFees(weiAmount, rateResult.reserve1, tradeInput.walletId));
        if (tradeInput.dest != TOMO_TOKEN_ADDRESS)
            require(feeBurnerContract.handleFees(weiAmount, rateResult.reserve2, tradeInput.walletId));
```

    - In function **doReserveTrade**:
        - Added balance checks.
        - Added **feeSharing** handle.
    - In function **validateTradeInput**:
        - Added parameter **isPayingFee** and allow **src** equals to **dest** if that's true.
- In *NetworkProxy.sol*:
    - Added function **addPayFeeCaller**.
    - Added function **payTxFee**.
- In *Reserve.sol*:
    - Renamed **tradeWithHint** to swap.
    - In function **trade**:
        - Added transfering fee to network.
    - Removed **tokenWallet** stuffs.
- In *WhiteList.sol*:
    - Removed **kgtToken** and related stuffs.

VULNERABILITIES FINDINGS

FIXED **MEDIUM**

UNSAFE/WRONG TRC20 FUNCTION CALL

Function **tradeFee** in **Network** contract contains unchecked TRC20 (ERC20) transfer:

```
      if (actualSrcAmount < tradeInput.srcAmount) {
          // if there is "change" send back to trader
          tradeInput.src.transfer(tradeInput.trader, (tradeInput.srcAmount -
actualSrcAmount));
      }
```

Similar problems are also found in function **tradeFee** in **WrapExpectedRate** and **WrapNetwork**:

```
      if (actualSrcAmount < tradeInput.srcAmount) {
          // if there is "change" send back to trader
          tradeInput.src.transfer(tradeInput.trader, (tradeInput.srcAmount -
actualSrcAmount));
      }
```

RECOMMENDED FIXES

- Wrap the calls with **require(…)**.

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.

FIXED LOW

GAS OPTIMIZATION IN FOR LOOP

Function **addFeeReserve** in **Network** contract contains a for loop to check if the reserve is added:

```
bool isReserveAdded = false;
for(uint i = 0; i < reserveArr.length; i++) {
  if (reserveArr[i] == address(reserve)) {
    isReserveAdded = true;
  }
}
```

The loop can be terminated early if the address is found.

Similar, another loop in **`listPairs`** can be terminated early:

```
  for (i = 0; i < reserveArr.length; i++) {
      if (reserve == reserveArr[i]) {
          if (add) {
              break; //already added
          } else {
              //remove
              reserveArr[i] = reserveArr[reserveArr.length - 1];
              reserveArr.length--;
          }
      }
  }
```

The fix has been applied in latest KyberNetwork contract.

RECOMMENDED FIXES

- Add *break* command:

```
bool isReserveAdded = false;
for (uint i = 0; i < reserveArr.length; i++) {
  if (reserveArr[i] == address(reserve)) {
    isReserveAdded = true;
    break;
  }
}
```

```
        for (i = 0; i < reserveArr.length; i++) {
            if (reserve == reserveArr[i]) {
                if (add) {
                    break; //already added
                } else {
                    //remove
                    reserveArr[i] = reserveArr[reserveArr.length - 1];
                    reserveArr.length--;
                    break;
                }
            }
        }
```

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.

FIXED LOW

UNNESSARY COSTLY OPERATION IN GETRESERVESX FUNCTIONS OF NETWORK

Function **getReservesPerTokenSrcCount** and **getReservesPerTokenDestCount** in **Network** contract contain unnessary array copy operator just to obtain the array length:

```
function getReservesPerTokenSrcCount(TRC20 token) public view returns(uint) {
  address[] memory reserveArr = reservesPerTokenSrc[token];
  return reserveArr.length;
}

function getReservesPerTokenDestCount(TRC20 token) public view returns(uint) {
  address[] memory reserveArr = reservesPerTokenDest[token];
  return reserveArr.length;
}
```

RECOMMENDED FIXES

- Change to:

```
function getReservesPerTokenSrcCount(TRC20 token) public view returns(uint) {
  return reservesPerTokenSrc[token].length;
}

function getReservesPerTokenDestCount(TRC20 token) public view returns(uint) {
  return reservesPerTokenDest[token].length;
}
```

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.

FIXED **LOW**

REDUNDANT CODE IN GETLISTEDTOKENATINDEX OF CONVERSIONRATES

In function **getListedTokenAtIndex** of **ConversionRates** contract:

```
function getListedTokensAtIndex(uint id) public view returns(TRC20) {
  require(id < listedTokens.length);
  return listedTokens[id];
}
```

The require call does not nesseray, Solidity already checks out-of-bound access for array.

RECOMMENDED FIXES

- Remove the **require** call.

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.

FIXED LOW

REDUNDANT MODIFIER IN HANDLEFEES OF FEESHARING

Function **handleFees** of **FeeSharing** contract does not contain any contract function call:

```
function handleFees(address wallet)
  public
  nonReentrant
  payable
  returns(bool)
{
    require(msg.sender == address(network));
    require(msg.value <= MAX_QTY);
    uint fee = msg.value;

    uint walletFee = fee * walletFeesInBps[wallet] / 10000;
    require(fee >= walletFee);

    if (walletFee > 0) {
        walletFeesToPay[wallet] += walletFee;
        emit AssignFeeToWallet(wallet, walletFee);
    }
    return true;
}
```

The **nonReentrant** modifier is not nesseray in this case.

RECOMMENDED FIXES

- Remove the **nonReentrant** modifier.

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.

FIXED **LOW**

INCONSISTENT COMMENTS REFERRING ETH ACROSS THE CODE

Many parts of the code contains comments/variable names referring ETH:

- **`LiquidityConversionRates`**:

```
            // ETH goes in, token goes out
```

- Network:

```
    /// @param token token address
    /// @param tomoToToken will it support ether to token trade
    /// @param tokenToTomo will it support token to ether trade
    /// @param add If true then list this pair, otherwise unlist it.
    function listPairForReserve(address reserve, TRC20 token, bool tomoToToken, bool
tokenToTomo, bool add)
…
    //@dev this function always src or dest are ether. can't do token to token
…
        //do the trade
        //src to ETH
…
    /// @notice use token address TOMO_TOKEN_ADDRESS for ether
    /// @dev do one trade with a reserve
…
            //this is for a "fake" trade when both src and dest are ethers.
…
        // reserve sends tokens/eth to network. network sends it to destination
```

- Utils:

```
    uint constant internal MAX_RATE  = (PRECISION * 10**6); // up to 1M tokens per ETH
```

RECOMMENDED FIXES

- Change the strings to Tomo.

12/04/2019: ChainTEX fixed this in branch **audit_v1_report** in commit *bea3872b43747d4f0e932aef1be2f4a6483e1005*.
16/04/2019: VeriChains found some other ETH references over the code, ChainTex updated the code within the same day and fixed at commit *d4387cb213e353d6b94c3e2c5aa5e81fc07f67fc*.

CONCLUSION

ChainTEX 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 ChainTEX smart contracts code. Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

ChainTEX handle the reported problems very carefully and handled all the reported problems.

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

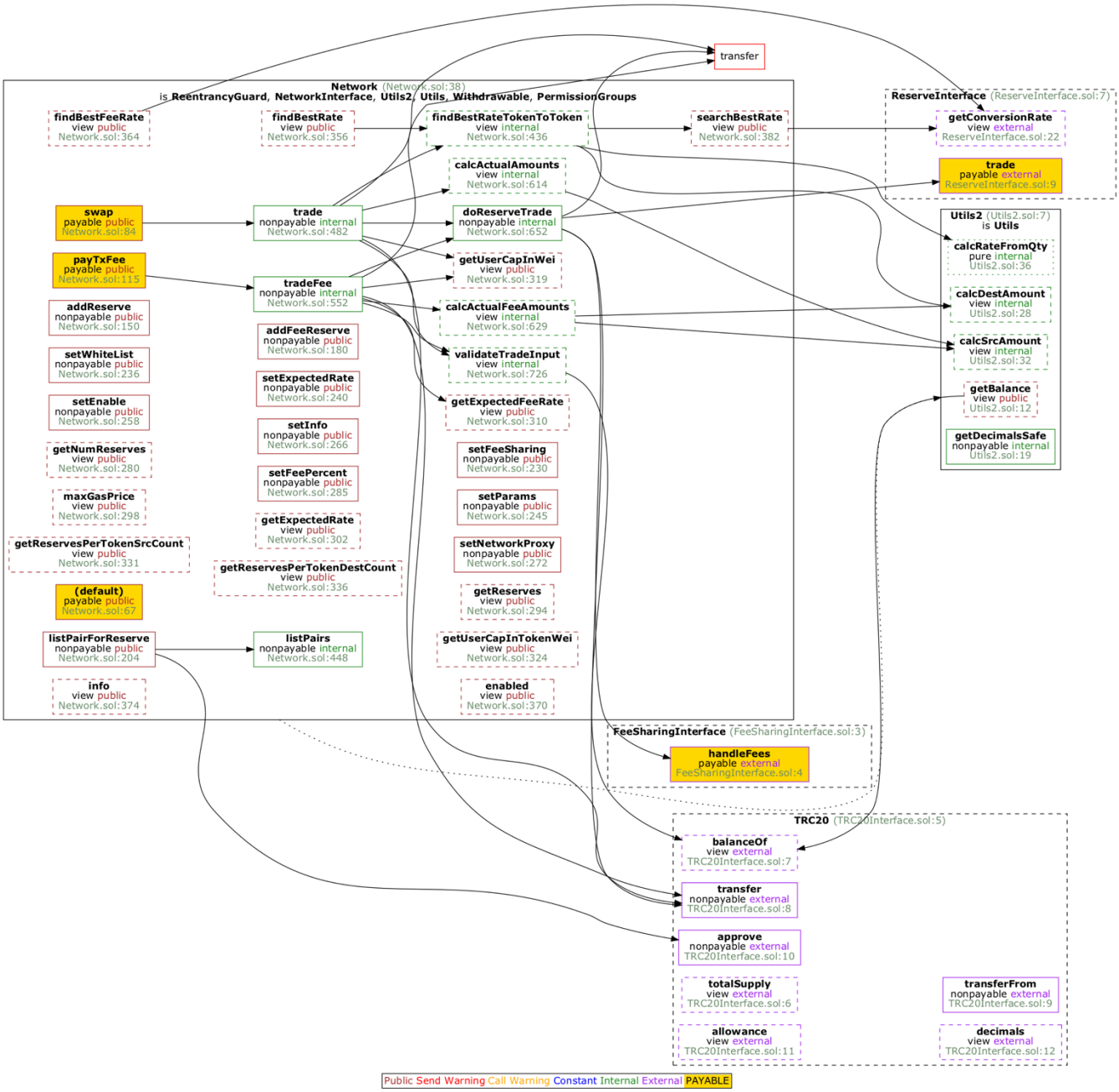


Figure 1 Network contract

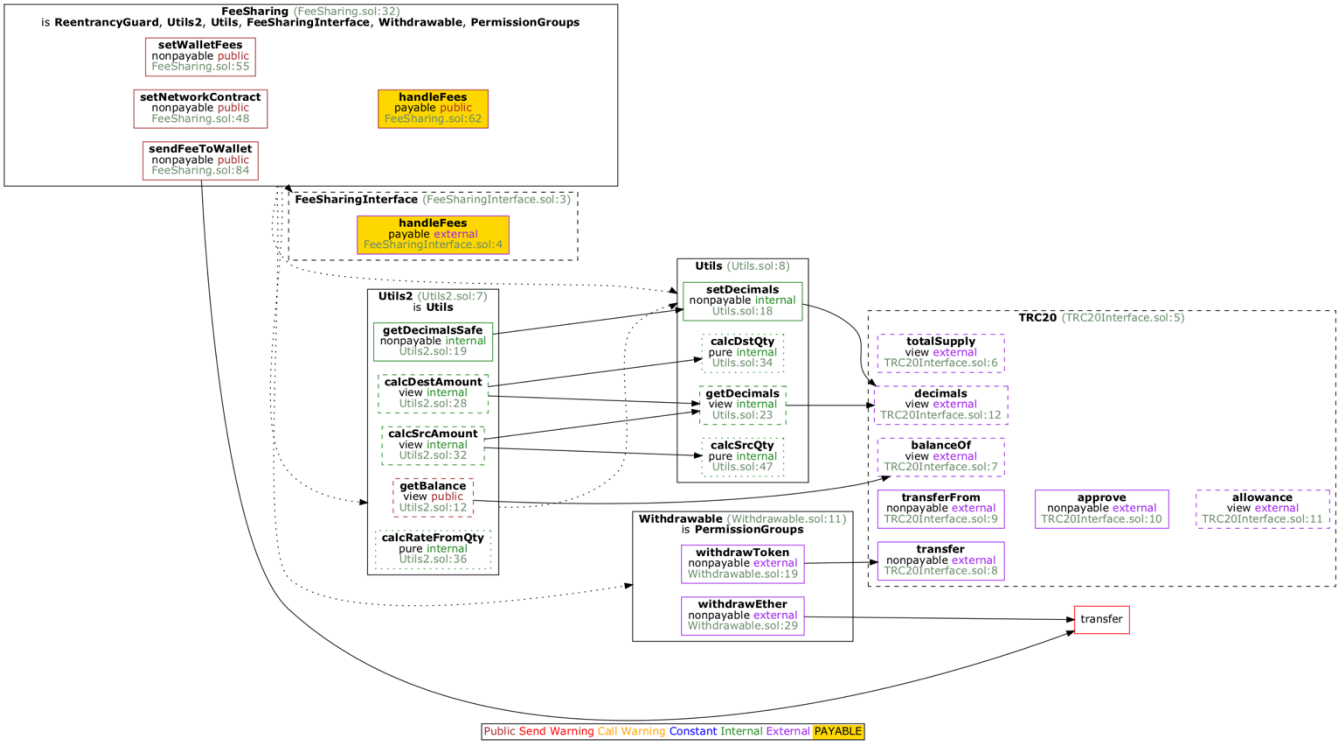


Figure 2 FeeSharing contract

APPENDIX II

ChainTEX Audit Modification 1

After receiving VeriChains Audit Report version 1, here are some modifications of the Smart Contracts.

- *Note*: All modifications can be found in the PR: https://github.com/chaintex/SmartContracts/pull/6/files from ***audit_v1_report*** branch to ***audit_v1*** branch

1. Modifying files

1. ConversionRates.sol 2. FeeSharing.sol
3. Network.sol
4. ExpectedRate.sol

5. Utils.sol
6. Withdrawable.sol
7. LiquidityConversionRates.sol 8. WhiteList.sol

and updated file tests.

2. Modifying Details

[Low] ConversionRates.sol - Redundant code

```
+++ b/contracts/ConversionRates.sol
@@ -321,7 +321,6 @@ contract ConversionRates is ConversionRatesInterface, VolumeImbalanceRecorder, U
     }

     function getListedTokensAtIndex(uint id) public view returns(TRC20) {
-      require(id < listedTokens.length);
       return listedTokens[id];
     }
```

Remove unnecessary check as Solidity already checks out-of-bound access for array.

[Low] FeeSharing.sol - Redundant modifier in handlFees of FeeSharing

```
+++ b/contracts/FeeSharing.sol
@@ -61,7 +61,6 @@ contract FeeSharing is Withdrawable, FeeSharingInterface, Utils2, ReentrancyGuar
     event AssignFeeToWallet(address wallet, uint walletFee);
     function handleFees(address wallet)
       public
-      nonReentrant
       payable
       returns(bool)
     {
```

Remove **nonReentrant** guard as it does not call any external contract.

[Medium] Network.sol - Unsafe TRC20 function call

```
@@ -586,8 +586,8 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
         expectedRate);

       if (actualSrcAmount < tradeInput.srcAmount) {
-          // if there is "change" send back to trader
-          tradeInput.src.transfer(tradeInput.trader, (tradeInput.srcAmount - actualSrcAmount));
+        // src is always a TRC20 token and not TOMO as we already checked if src is TOMO and return above
+        require(tradeInput.src.transfer(tradeInput.trader, (tradeInput.srcAmount - actualSrcAmount)));
       }
```

Add **require()** to check TRC20 transfer.

[Low] Network.sol - Gas optimization for loop

```
       address[] memory reserveArr = reservesPerTokenDest[token];
@@ -185,6 +185,7 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
       for(uint i = 0; i < reserveArr.length; i++) {
         if (reserveArr[i] == address(reserve)) {
           isReserveAdded = true;
+          break;
         }
       }
```

```
@@ -461,6 +460,7 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
                 //remove
                 reserveArr[i] = reserveArr[reserveArr.length - 1];
                 reserveArr.length--;
+                break;
               }
             }
           }
```

Break earlier in **addReserveFee** and **listPairs** functions.

[Low] Network.sol - Redundant codes

```
@@ -329,13 +330,11 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
     }

     function getReservesPerTokenSrcCount(TRC20 token) public view returns(uint) {
-      address[] memory reserveArr = reservesPerTokenSrc[token];
-      return reserveArr.length;
+      return reservesPerTokenSrc[token].length;
     }

     function getReservesPerTokenDestCount(TRC20 token) public view returns(uint) {
-      address[] memory reserveArr = reservesPerTokenDest[token];
-      return reserveArr.length;
+      return reservesPerTokenDest[token].length;
     }
```

Remove unnecessary arry copy operator in **getReservesPerTokenSrcCount** and **getReservesPerTokenDestCount**

[Low] Network.sol - Add one more check for balance when calling handleFees()

```
@@ -711,8 +711,11 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {

         if (feeSharing != address(0) && feeInWei > 0) {
           require(address(this).balance >= feeInWei);
+          expectedTomoBal -= feeInWei;
           // transfer fee to feeSharing
           require(feeSharing.handleFees.value(feeInWei)(walletId));
+          // expected only fee in wei is transfered to feeSharing
+          require(address(this).balance == expectedTomoBal);
         }
```

Add one more check for balance after calling **feeSharing.handleFees()** function to make sure balance is deducted correctly.

[Low] Network.sol - Change modifier onlyAdmin to onlyOperator

```
@@ -143,11 +143,11 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {

     event AddReserveToNetwork(ReserveInterface reserve, bool add);

-    /// @notice can be called only by admin
+    /// @notice can be called only by operator
     /// @dev add or deletes a reserve to/from the network.
     /// @param reserve The reserve address.
     /// @param add If true, the add reserve. Otherwise delete reserve.
-    function addReserve(ReserveInterface reserve, bool add) public onlyAdmin {
+    function addReserve(ReserveInterface reserve, bool add) public onlyOperator {
@@ -169,7 +169,7 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
         }
     }

-    // @notice can be called only by admin
+    // @notice can be called only by operator
     // @dev add or delete a reserve for fee to/from the network
     // @dev will need to call separately function addReserve
     // @dev this reserve must list pair (Tomo, token) to support trade from Tomo -> token
@@ -177,7 +177,7 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
     // @param token: token to map with the reserve

     event AddFeeReserveToNetwork(ReserveInterface reserve, TRC20 token);
-    function addFeeReserve(ReserveInterface reserve, TRC20 token) public onlyAdmin {
+    function addFeeReserve(ReserveInterface reserve, TRC20 token) public onlyOperator {
```

```
@@ -194,15 +195,15 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {

     event ListReservePairs(address reserve, TRC20 src, TRC20 dest, bool add);

-    /// @notice can be called only by admin
+    /// @notice can be called only by operator
     /// @dev allow or prevent a specific reserve to trade a pair of tokens
     /// @param reserve The reserve address.
     /// @param token token address
-    /// @param tomoToToken will it support ether to token trade
-    /// @param tokenToTomo will it support token to ether trade
+    /// @param tomoToToken will it support tomo to token trade
+    /// @param tokenToTomo will it support token to tomo trade
     /// @param add If true then list this pair, otherwise unlist it.
     function listPairForReserve(address reserve, TRC20 token, bool tomoToToken, bool tokenToTomo, bool add)
-        public onlyAdmin
+        public onlyOperator
     {
         require(isReserve[reserve]);
```

Using **onlyOperator** instead of **onlyAdmin** (similar to current **KyberNetwork.sol** contract) for **addReserve, addReserveFee, listPairForReserve** functions.

[Low] Inconsistent comments/params referring to ETH ExpectedRate.sol; LiquidityConversionRates.sol; Network.sol;

Utils.sol; WhiteList.sol; Withdrawable.sol

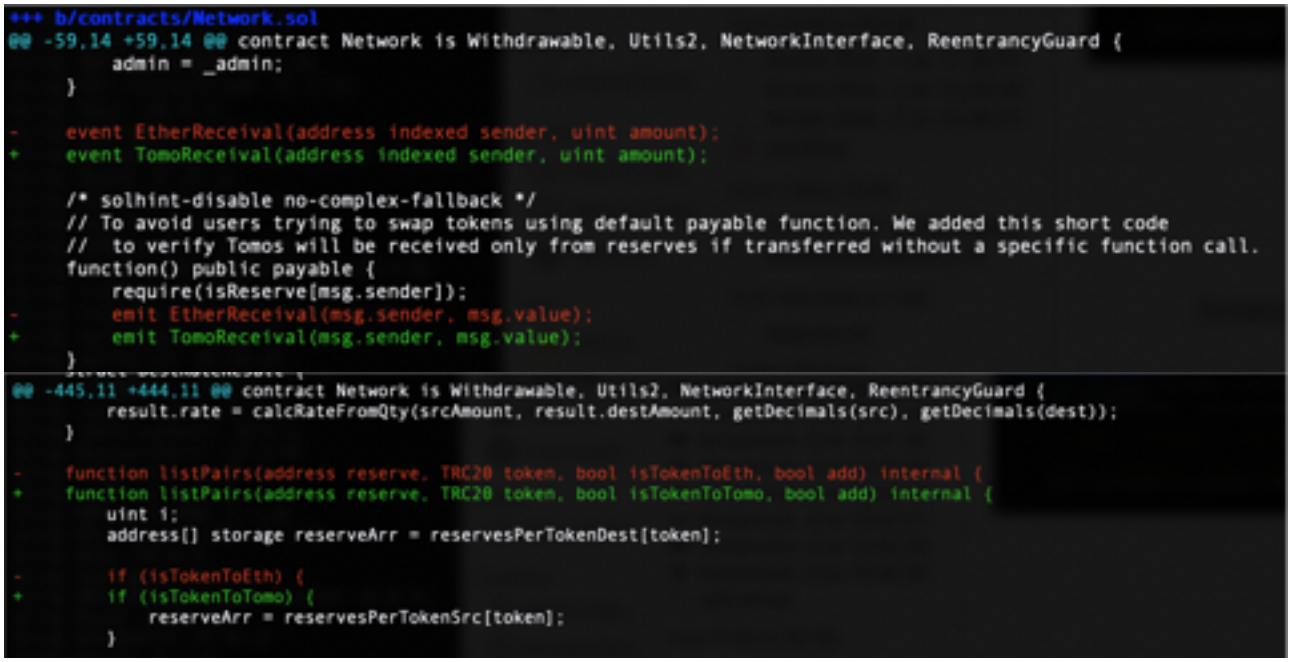
```
+++ b/contracts/ExpectedRate.sol
@@ -109,9 +109,9 @@ contract ExpectedRate is Withdrawable, ExpectedRateInterface, Utils2 {
         uint rateTomoToDest;
         (reserve, rateSrcToTomo) = network.searchBestRate(src, TOMO_TOKEN_ADDRESS, srcQty);

-        uint ethQty = calcDestAmount(src, TOMO_TOKEN_ADDRESS, srcQty, rateSrcToTomo);
+        uint tomoQty = calcDestAmount(src, TOMO_TOKEN_ADDRESS, srcQty, rateSrcToTomo);

-        (reserve, rateTomoToDest) = network.searchBestRate(TOMO_TOKEN_ADDRESS, dest, ethQty);
+        (reserve, rateTomoToDest) = network.searchBestRate(TOMO_TOKEN_ADDRESS, dest, tomoQty);
         return rateSrcToTomo * rateTomoToDest / PRECISION;
     }
 }
```

```
+++ b/contracts/LiquidityConversionRates.sol
@@ -160,7 +160,7 @@ contract LiquidityConversionRates is ConversionRatesInterface, LiquidityFormula,
         if (conversionToken != token) return 0;

         if (buy) {
-            // ETH goes in, token goes out
+            // TOMO goes in, token goes out
             deltaEInFp = fromWeiToFp(qtyInSrcWei);
             if (deltaEInFp > maxEthCapBuyInFp) return 0;
```



```
+++ b/contracts/Network.sol
@@ -59,14 +59,14 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
         admin = _admin;
     }

-    event EtherReceival(address indexed sender, uint amount);
+    event TomoReceival(address indexed sender, uint amount);

     /* solhint-disable no-complex-fallback */
     // To avoid users trying to swap tokens using default payable function. We added this short code
     //  to verify Tomos will be received only from reserves if transferred without a specific function call.
     function() public payable {
         require(isReserve[msg.sender]);
-        emit EtherReceival(msg.sender, msg.value);
+        emit TomoReceival(msg.sender, msg.value);
     }
@@ -445,11 +444,11 @@ contract Network is Withdrawable, Utils2, NetworkInterface, ReentrancyGuard {
         result.rate = calcRateFromQty(srcAmount, result.destAmount, getDecimals(src), getDecimals(dest));
     }

-    function listPairs(address reserve, TRC20 token, bool isTokenToEth, bool add) internal {
+    function listPairs(address reserve, TRC20 token, bool isTokenToTomo, bool add) internal {
         uint i;
         address[] storage reserveArr = reservesPerTokenDest[token];

-        if (isTokenToEth) {
+        if (isTokenToTomo) {
             reserveArr = reservesPerTokenSrc[token];
         }
```