QURAS BLOCKCHAIN
Audit Report

Conducted in Jan 2020

1. Executive Summary

QURAS is a secret contract platform that fulfills privacy protection needs. 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 issues from QURAS team. There were multiple low level issues and note level issues. QURAS team responded to the issues and TECHFUND verified the source code again on Github to make sure that the issues were fixe.

2. Scope

i . Windows build (.exe)

We checked the dll and exe files generated for monkey testing and possible scenarios like DLL hijacking, strings exposed, files being accessed during run time, network interactions and so on.

ii . Blockchain Source Code

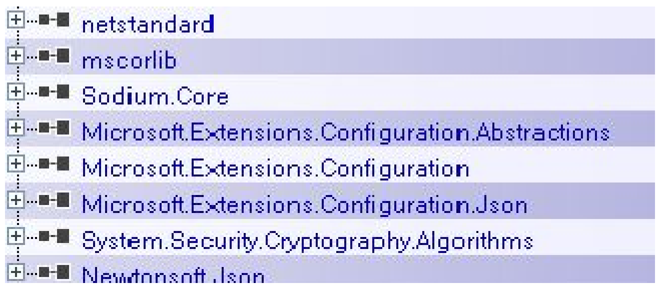
The target was provided by the QURAS team as a repository https://github.com/quras-official/ . The code includes several third party dependencies like libSodium, levelDB and so on.

iii . API Source Code

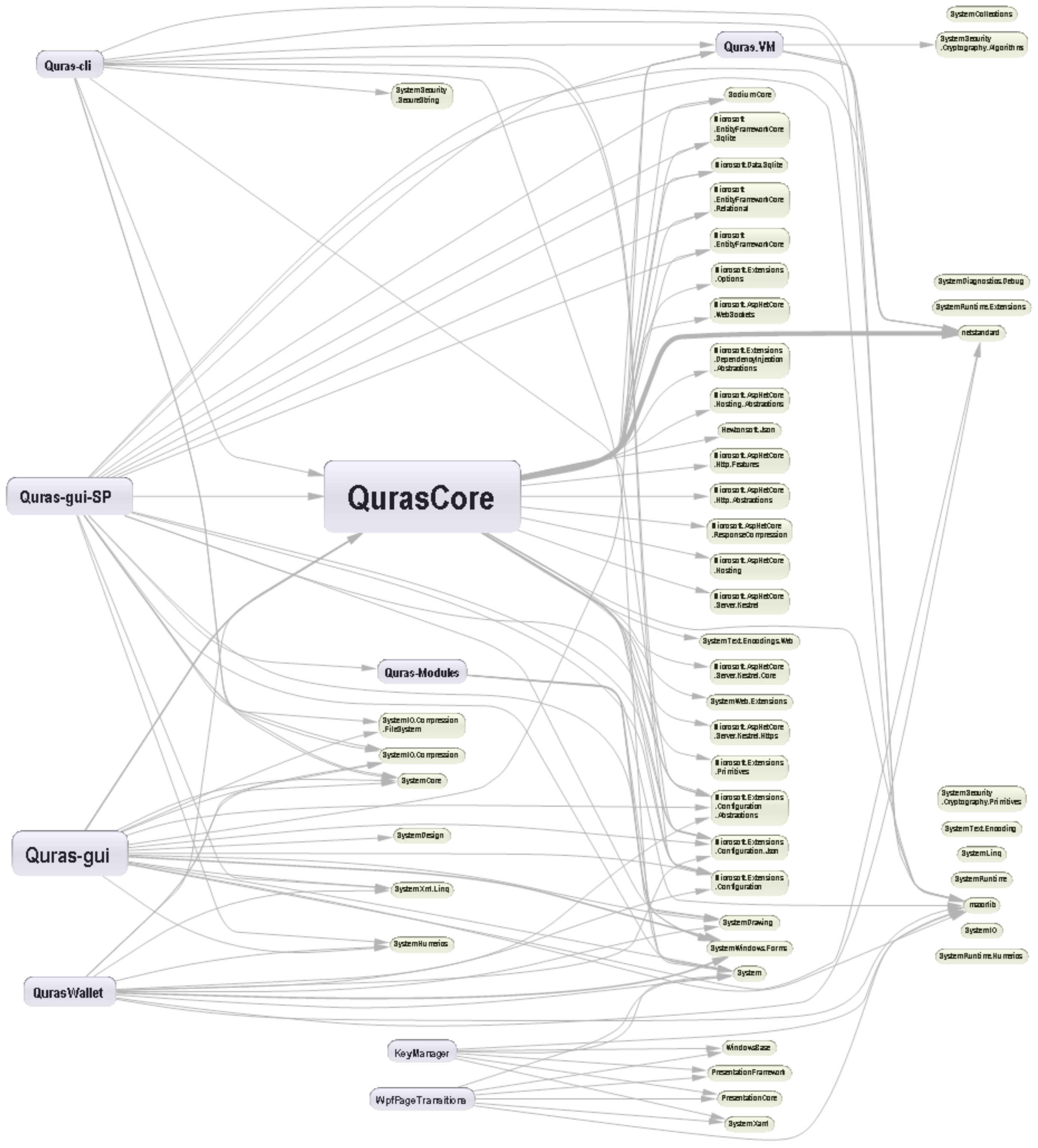
We also tested the API source code provided by the QURAS team as repositories https://github.com/quras-official/quras-nodejs-api , https://github.com/quras-official/quras-backend-service . We performed security and performance tests on the endpoints to find bottleneck APIs and security of the whole system.

3. Source Summary

Quras-cli , Quaras-GUI and Quras Wallet have direct dependency on QurasCore.



netstandard
mscorlib
Sodium.Core
Microsoft.Extensions.Configuration.Abstractions
Microsoft.Extensions.Configuration
Microsoft.Extensions.Configuration.Json
System.Security.Cryptography.Algorithms
Newtonsoft.Json


Quras-cli
Quras.VM
System.Collections
System.Security.Cryptography.Algorithms
System.Security.SecureString
Sodium.Core
Microsoft.EntityFrameworkCore.Sqlite
Microsoft.Data.Sqlite
Microsoft.EntityFrameworkCore.Relational
Microsoft.EntityFrameworkCore
Microsoft.Extensions.Options
Microsoft.AspNetCore.WebSockets
System.Diagnostics.Debug
System.Runtime.Extensions
netstandard
Microsoft.Extensions.DependencyInjection.Abstractions
Microsoft.AspNetCore.Hosting.Abstractions
Newtonsoft.Json
Quras-gui-SP
QurasCore
Microsoft.AspNetCore.Http.Features
Microsoft.AspNetCore.Http.Abstractions
Microsoft.AspNetCore.ResponseCompression
Microsoft.AspNetCore.Hosting
Microsoft.AspNetCore.Server.Kestrel
System.Text.Encodings.Web
Quras-Modules
Microsoft.AspNetCore.Server.Kestrel.Core
System.Web.Extensions
System.IO.Compression.FileSystem
Microsoft.AspNetCore.Server.Kestrel.Https
System.IO.Compression
System.Core
Microsoft.Extensions.Primitives
Microsoft.Extensions.Configuration.Abstractions
System.Security.Cryptography.Primitives
System.Text.Encoding
System.Design
Quras-gui
Microsoft.Extensions.Configuration.Json
System.Linq
System.Runtime
Microsoft.Extensions.Configuration
System.Xml.Linq
mscorlib
System.Drawing
System.IO
System.Windows.Forms
System.Numerics
System.Runtime.Numerics
System
QurasWallet
KeyManager
WindowsBase
PresentationFramework
WpfPageTransitions
PresentationCore
System.Xaml

4. Issues and descriptions

Detailed Analysis and Suggestions [API Source code]

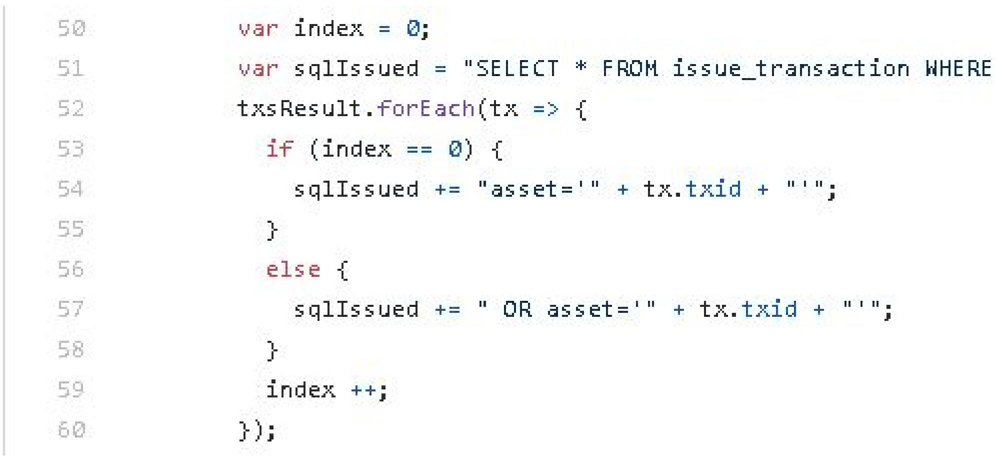
These are the results of detailed analysis and suggestions derived from it. There are a total of 3 findings for API server.

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

4.1. Potential SQL Injection

Risk Level: Middle

It could be noted at many places SQL queries are generated on the fly, although the input is from already stored IDs but another process that has access to local levelDb will be able to execute SQL injection with access rights of the QURAS Blockchain`s process user.



```
50          var index = 0;
51          var sqlIssued = "SELECT * FROM issue_transaction WHERE
52          txsResult.forEach(tx => {
53            if (index == 0) {
54              sqlIssued += "asset='" + tx.txid + "'";
55            }
56            else {
57              sqlIssued += " OR asset='" + tx.txid + "'";
58            }
59            index ++;
60          });
```

It is suggested to use **connection.escape** while handling any query generation.

api-service > routes > v1 > address > 126
api-service > routes > v1 > addresses > 87
api-service > routes > v1 > assets > 54
api-service > routes > v1 > txs > 89

Fixed
The issue was handled using mysql escape function provided by the library to prevent any potential injection to happen. TECHFUND verified that all queries generated on the fly were handling variables properly and no potential case of injection is now possible.

4.2. Static content served via API server

Risk Level: None

API server is being used to serve static content, NodeJS is known to be bad performing to server static content as compared to Nginx. It is suggested to use a static server in production else it can lead to server performance bottleneck.

Fixed
TECHFUND verified that proper Nginx configuration is now being used to prevent any static content being served by Node.JS server.

4.3. SQL Connenction bottleneck

Risk Level: Note

There is a significant perfomance degradation of the APIs because of repetitive connection handshakes multiple times. It is suggested to create a `**single**` connection Pool during server startup and share that across the platform, instead of connecting to the database multiple times.

Fixed
TECHFUND verified that connection pools are being used to handle SQL connection and APIs are no longer creating multiple connection requests to the database server.

Detailed Analysis and Suggestions [QURAS Blockchain / EXEs - DLLs]

These are the results of detailed analysis and suggestions derived from it. There are a total of 3 findings for API server.

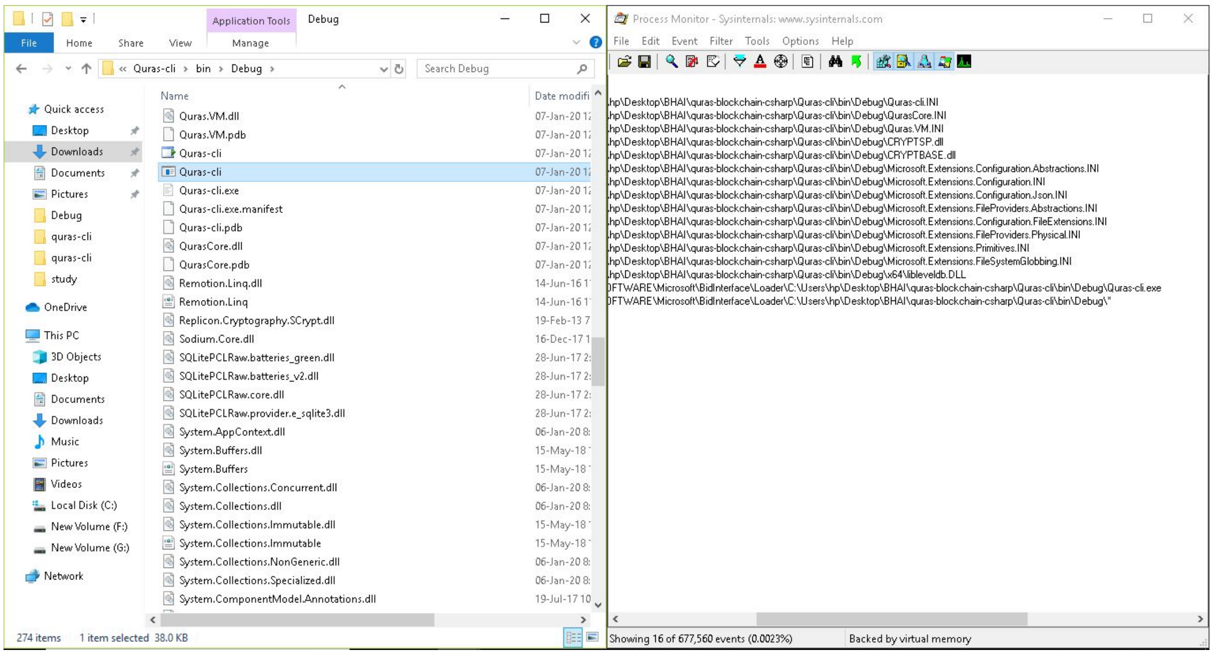
Risk Level: Critical	0
Risk Level: High	1
Risk Level: Middle	5
Risk Level: Low	0
Risk Level: Note	5
Risk Level: None	2

4.4. DLL High jacking

Risk Level: High

Dynamic-link library (DLL) side-loading is a popular cyber attack method that takes advantage of how Microsoft Windows applications handle DLL files.
Windows allows applications to load DLLs at runtime. Applications can specify the location of DLLs to load by specifying a **full path**, using DLL redirection, or by using a manifest. If none of these methods are used, Windows attempts to locate the DLL by searching a predefined set of directories in a set order.
It is possible for other applications to sideload DLL or corrupt the application by providing a corrupted DLL higher in the search from Windows OS.

We noticed it is possible to sideload DLLs in QURAS GUI and QURAS cli application. As a fix QURAS is suggested to check for DLL integrity and fix the location to prevent side loading.



```
Name                                        Date modifi
Quras.VM.dll                                07-Jan-20 1
Quras.VM.pdb                                07-Jan-20 1
Quras-cli                                   07-Jan-20 1
Quras-cli                                   07-Jan-20 1
Quras-cli.exe                               07-Jan-20 1
Quras-cli.exe.manifest                      07-Jan-20 1
Quras-cli.pdb                               07-Jan-20 1
QurasCore.dll                               07-Jan-20 1
QurasCore.pdb                               07-Jan-20 1
Remotion.Linq.dll                           14-Jun-16 1
Remotion.Linq                               14-Jun-16 1
Replicon.Cryptography.SCrypt.dll            19-Feb-13 7
Sodium.Core.dll                             16-Dec-17 1
SQLitePCLRaw.batteries_green.dll            28-Jun-17 2
SQLitePCLRaw.batteries_v2.dll               28-Jun-17 2
SQLitePCLRaw.core.dll                       28-Jun-17 2
SQLitePCLRaw.provider.e_sqlite3.dll         28-Jun-17 2
System.AppContext.dll                       06-Jan-20 8
System.Buffers.dll                          15-May-18
System.Buffers                              15-May-18
System.Collections.Concurrent.dll           06-Jan-20 8
System.Collections.dll                      06-Jan-20 8
System.Collections.Immutable.dll            15-May-18
System.Collections.Immutable                15-May-18
System.Collections.NonGeneric.dll           06-Jan-20 8
System.Collections.Specialized.dll          06-Jan-20 8
System.ComponentModel.Annotations.dll       19-Jul-17 10

274 items    1 item selected  38.0 KB
```

```
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Quras-cli.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\QurasCore.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Quras.VM.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\CRYPTSP.dll
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\CRYPTBASE.dll
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.Configuration.Abstractions.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.Configuration.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.Configuration.Json.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.FileProviders.Abstractions.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.Configuration.FileExtensions.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.FileProviders.Physical.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.Primitives.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Microsoft.Extensions.FileSystemGlobbing.INI
hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\x64\libleveldb.DLL
DFTWARE\Microsoft\BidInterface\Loader\C:\Users\hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\Quras-cli.exe
DFTWARE\Microsoft\BidInterface\Loader\C:\Users\hp\Desktop\BHAI\quras-blockchain-csharp\Quras-cli\bin\Debug\*

Showing 16 of 677,560 events (0.0023%)    Backed by virtual memory
```

Fixed

TECHFUND was pleased to see that proper signature verification is now in place to decrease the effect of this issue and DLL side-loading will not be an issue for the QURAS binary.

4.5. Vulnerable Randomness

Risk Level: Middle

```
                        boost::winapi::CRYPT_VERIFYCONTEXT_ | boost::winapi::
            {
                random_ = 0;
            }
#else
            random_ = std::fopen( "/dev/urandom", "rb" );
#endif

            std::memset(rd_, 0, sizeof(rd_));
        }

        ~seed_rng() BOOST_NOEXCEPT
        {
            if (random_) {
```

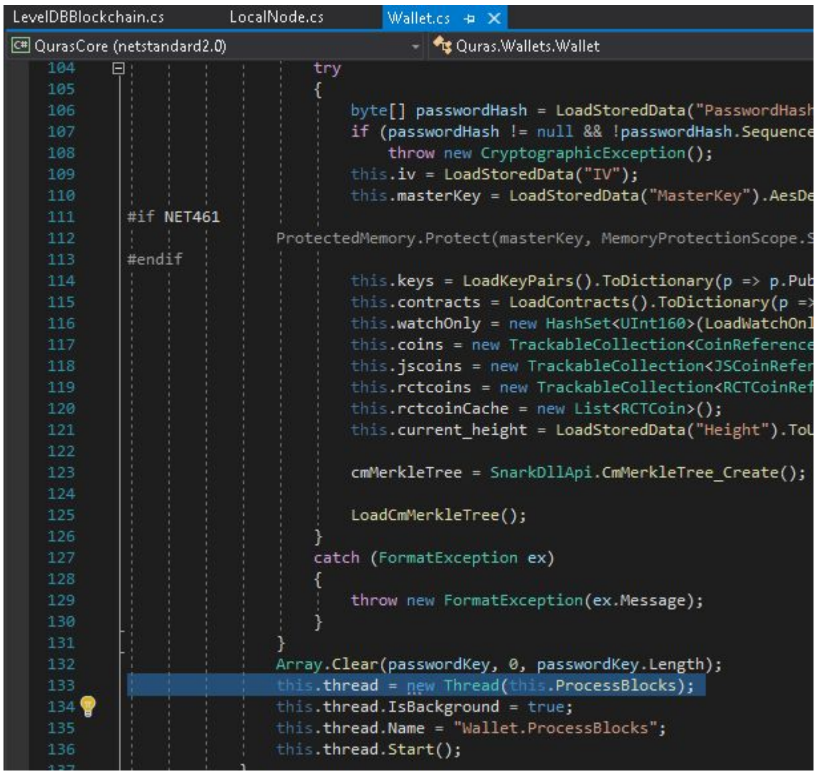
/dev/urandom is an insecure way of generating random string and has been known to cause issues in the past. It is suggested to use more secure random number generators as compared to the present method. (https://nvd.nist.gov/vuln/detail/CVE-2003-0094)

quras-anonymous-library > Common > include > boost > uuid > detail > seed_rng.hpp

Fixed
TECHFUND verified that weak random number generator are now not being used and the issue has been fixed by the QURAS team.

4.6. Prevent creating threads explicitly

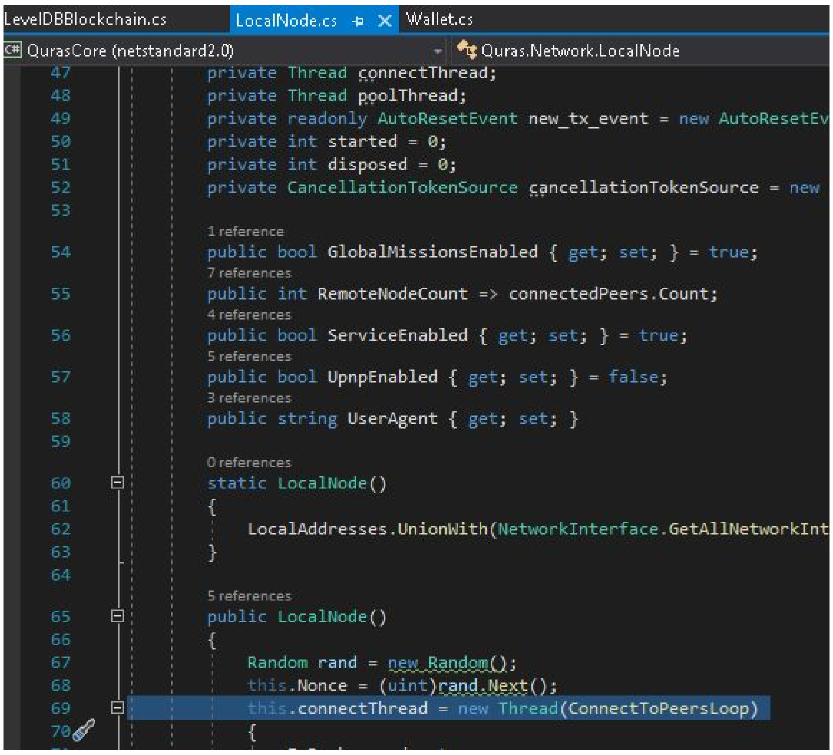
Risk Level: Note



LevelDBBlockchain.cs   LocalNode.cs   Wallet.cs

QurasCore (netstandard2.0)   Quras.Wallets.Wallet

```
                try
                {
                    byte[] passwordHash = LoadStoredData("PasswordHash
                    if (passwordHash != null && !passwordHash.Sequence
                        throw new CryptographicException();
                    this.iv = LoadStoredData("IV");
                    this.masterKey = LoadStoredData("MasterKey").AesDe
#if NET461
            ProtectedMemory.Protect(masterKey, MemoryProtectionScope.S
#endif
                    this.keys = LoadKeyPairs().ToDictionary(p => p.Pub
                    this.contracts = LoadContracts().ToDictionary(p =>
                    this.watchOnly = new HashSet<UInt160>(LoadWatchOnl
                    this.coins = new TrackableCollection<CoinReference
                    this.jscoins = new TrackableCollection<JSCoinRefer
                    this.rctcoins = new TrackableCollection<RCTCoinRef
                    this.rctcoinCache = new List<RCTCoin>();
                    this.current_height = LoadStoredData("Height").ToU

                    cmMerkleTree = SnarkDllApi.CmMerkleTree_Create();

                    LoadCmMerkleTree();
                }
                catch (FormatException ex)
                {
                    throw new FormatException(ex.Message);
                }
            }
            Array.Clear(passwordKey, 0, passwordKey.Length);
            this.thread = new Thread(this.ProcessBlocks);
            this.thread.IsBackground = true;
            this.thread.Name = "Wallet.ProcessBlocks";
            this.thread.Start();
```



```
        private Thread connectThread;
        private Thread poolThread;
        private readonly AutoResetEvent new_tx_event = new AutoResetEv
        private int started = 0;
        private int disposed = 0;
        private CancellationTokenSource cancellationTokenSource = new

        1 reference
        public bool GlobalMissionsEnabled { get; set; } = true;
        7 references
        public int RemoteNodeCount => connectedPeers.Count;
        4 references
        public bool ServiceEnabled { get; set; } = true;
        5 references
        public bool UpnpEnabled { get; set; } = false;
        3 references
        public string UserAgent { get; set; }

        0 references
        static LocalNode()
        {
            LocalAddresses.UnionWith(NetworkInterface.GetAllNetworkInt
        }

        5 references
        public LocalNode()
        {
            Random rand = new Random();
            this.Nonce = (uint)rand.Next();
            this.connectThread = new Thread(ConnectToPeersLoop)
            {
```

Fixed

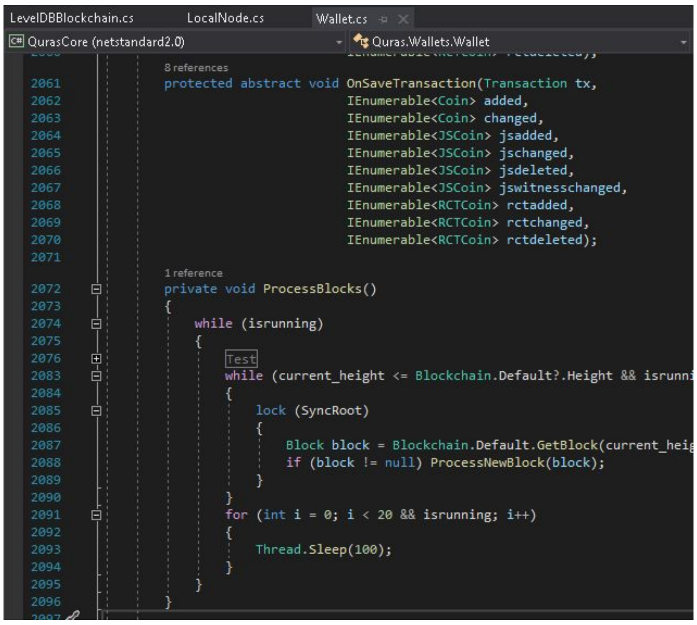
TECHFUND suggested a couple of methods and QURAS team implemented an overall memory Stack limit on the processes handled by newly created threads to make sure the binary does not behave abruptly in systems with low system configuration. TECHFUND finds the fix satisfactory to handle the situation.

4.7. Prevent sleeping in the thread

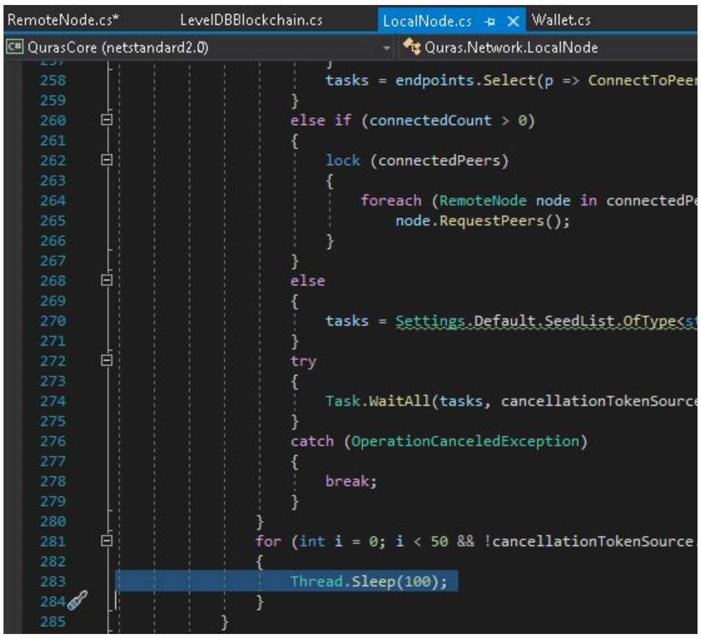
Risk Level: Note

Threads are a limited resource, they take approximately 200,000 cycles to create and about 100,000 cycles to destroy. By default they reserve 1 megabyte of virtual memory for its stack and use 2,000-8,000 cycles for each context switch. This makes any waiting thread a *huge* waste.

It is strongly suggested to *not* use thread.sleep in the code and switch to Task Parallel library.



```
LevelDBBlockchain.cs    LocalNode.cs    Wallet.cs
QurasCore (netstandard2.0)    Quras.Wallets.Wallet

        8 references
        protected abstract void OnSaveTransaction(Transaction tx,
                                                IEnumerable<Coin> added,
                                                IEnumerable<Coin> changed,
                                                IEnumerable<JSCoin> jsadded,
                                                IEnumerable<JSCoin> jschanged,
                                                IEnumerable<JSCoin> jsdeleted,
                                                IEnumerable<JSCoin> jswitnesschanged,
                                                IEnumerable<RCTCoin> rctadded,
                                                IEnumerable<RCTCoin> rctchanged,
                                                IEnumerable<RCTCoin> rctdeleted);

        1 reference
        private void ProcessBlocks()
        {
            while (isrunning)
            {
                Test
                while (current_height <= Blockchain.Default?.Height && isrunni
                {
                    lock (SyncRoot)
                    {
                        Block block = Blockchain.Default.GetBlock(current_hei
                        if (block != null) ProcessNewBlock(block);
                    }
                }
                for (int i = 0; i < 20 && isrunning; i++)
                {
                    Thread.Sleep(100);
                }
            }
        }
```



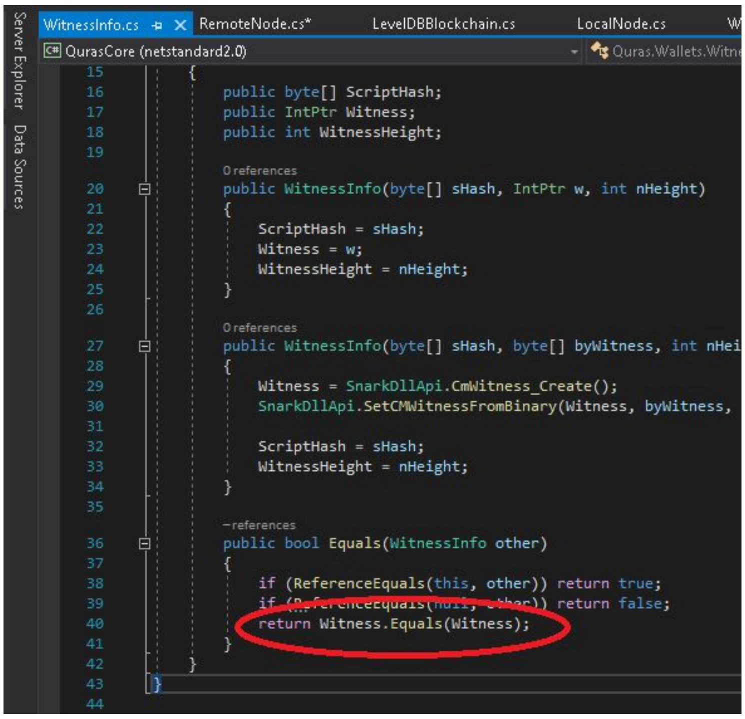
```
RemoteNode.cs*    LevelDBBlockchain.cs    LocalNode.cs    Wallet.cs
QurasCore (netstandard2.0)    Quras.Network.LocalNode

                    tasks = endpoints.Select(p => ConnectToPeer
                }
                else if (connectedCount > 0)
                {
                    lock (connectedPeers)
                    {
                        foreach (RemoteNode node in connectedPe
                            node.RequestPeers();
                    }
                }
                else
                {
                    tasks = Settings.Default.SeedList.OfType<s
                }
                try
                {
                    Task.WaitAll(tasks, cancellationTokenSource
                }
                catch (OperationCanceledException)
                {
                    break;
                }
            }
            for (int i = 0; i < 50 && !cancellationTokenSource.
            {
                Thread.Sleep(100);
            }
        }
```

Fixed
TECHFUND verifies that the QURAS team handled the thread sleeping and moved to a more robust async-await / delay model.

4.8. Buggy comparison in wallet witness

Risk Level: Middle

Identical expressions should not be used on both sides of a binary operator. It is wrong by design and should not be used in code to return a TRUE value. This will lead to completely wrong equality of Witnessifo.



```
    {
        public byte[] ScriptHash;
        public IntPtr Witness;
        public int WitnessHeight;

        0 references
        public WitnessInfo(byte[] sHash, IntPtr w, int nHeight)
        {
            ScriptHash = sHash;
            Witness = w;
            WitnessHeight = nHeight;
        }

        0 references
        public WitnessInfo(byte[] sHash, byte[] byWitness, int nHei
        {
            Witness = SnarkDllApi.CmWitness_Create();
            SnarkDllApi.SetCMWitnessFromBinary(Witness, byWitness,

            ScriptHash = sHash;
            WitnessHeight = nHeight;
        }

        -references
        public bool Equals(WitnessInfo other)
        {
            if (ReferenceEquals(this, other)) return true;
            if (ReferenceEquals(null, other)) return false;
            return Witness.Equals(Witness);
        }
    }
}
```

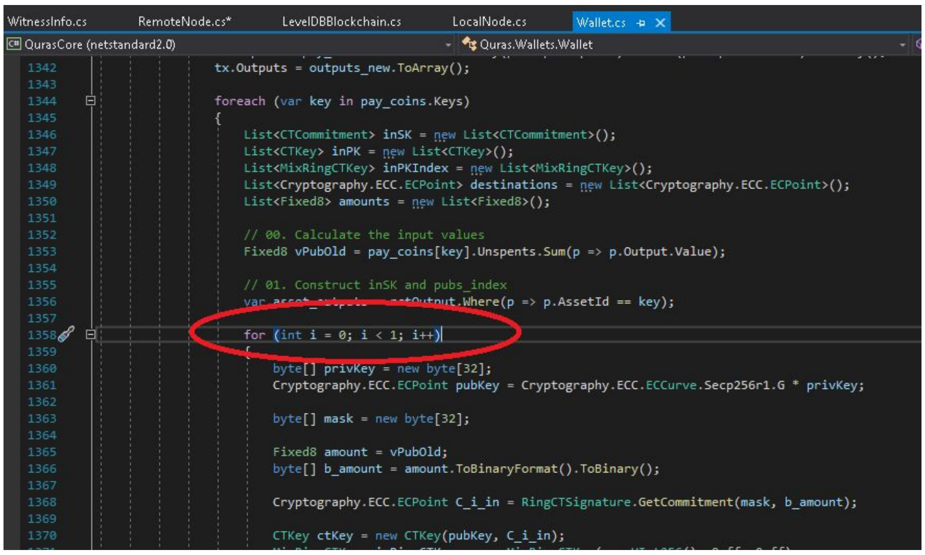
Fixed

TECHFUND verifies that this potential misleading code has been fixed and a clearer function equality is implemented. This should help developers understand code in a better way and prevent any potential issues.

4.9. Non required for loop

Risk Level: None

It is suggested to prevent “for loops” just to run a piece of code in a normal way. If this was not intended by QURAS, it should be fixed from code sanity point of view.



```
tx.Outputs = outputs_new.ToArray();

foreach (var key in pay_coins.Keys)
{
    List<CTCommitment> inSK = new List<CTCommitment>();
    List<CTKey> inPK = new List<CTKey>();
    List<MixRingCTKey> inPKIndex = new List<MixRingCTKey>();
    List<Cryptography.ECC.ECPoint> destinations = new List<Cryptography.ECC.ECPoint>();
    List<Fixed8> amounts = new List<Fixed8>();

    // 00. Calculate the input values
    Fixed8 vPubOld = pay_coins[key].Unspents.Sum(p => p.Output.Value);

    // 01. Construct inSK and pubs_index
    var asset_outputs = [illegible]tOutput.Where(p => p.AssetId == key);

    for (int i = 0; i < 1; i++)
    {
        byte[] privKey = new byte[32];
        Cryptography.ECC.ECPoint pubKey = Cryptography.ECC.ECCurve.Secp256r1.G * privKey;

        byte[] mask = new byte[32];

        Fixed8 amount = vPubOld;
        byte[] b_amount = amount.ToBinaryFormat().ToBinary();

        Cryptography.ECC.ECPoint C_i_in = RingCTSignature.GetCommitment(mask, b_amount);

        CTKey ctKey = new CTKey(pubKey, C_i_in);
```

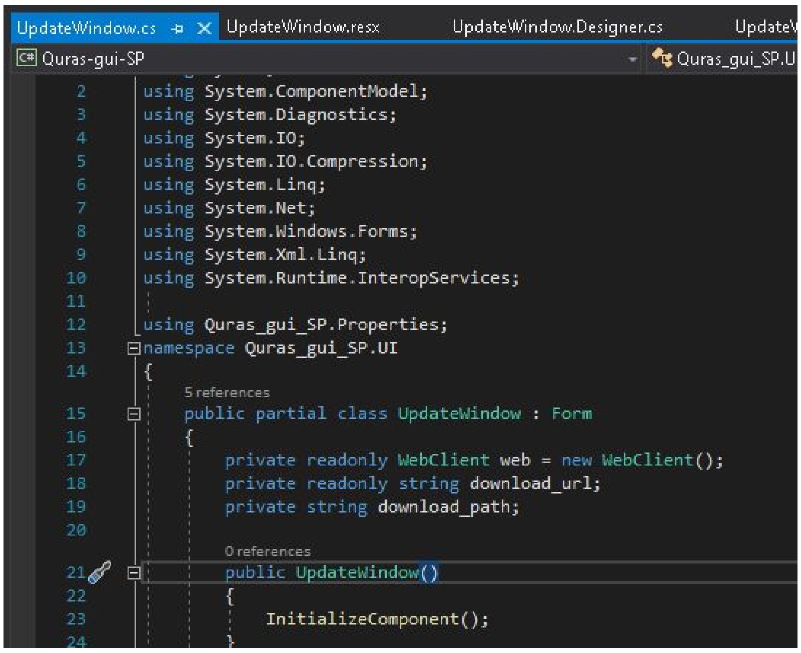
Fixed

TECHFUND verified that the code has been fixed and ambiguity has been removed.

4.10. WebClient should be disposed

Risk Level: Note

The operating system can only handle having so many sockets (. WebClient) open at any given time. Therefore, it is important to Dispose of them as soon as they are no longer needed, rather than relying on the garbage collector to call these objects' finalizers at some nondeterministic point in the future.



```
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.IO.Compression;
using System.Linq;
using System.Net;
using System.Windows.Forms;
using System.Xml.Linq;
using System.Runtime.InteropServices;

using Quras_gui_SP.Properties;
namespace Quras_gui_SP.UI
{
    5 references
    public partial class UpdateWindow : Form
    {
        private readonly WebClient web = new WebClient();
        private readonly string download_url;
        private string download_path;

        0 references
        public UpdateWindow()
        {
            InitializeComponent();
        }
```

Similar behaviour should be corrected in

QurasCore > Core > InvocationTransaction.cs

QurasWallet > Cryptography > CertificateQueryService.cs

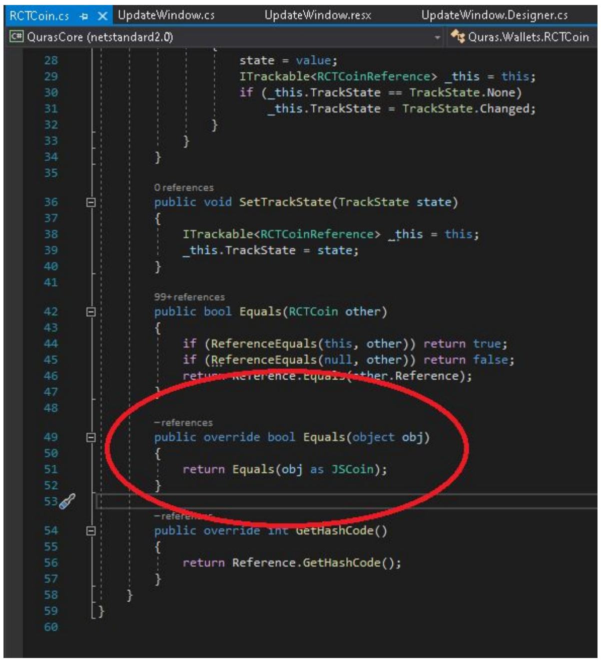
Fixed

TECHFUND verified that now proper handling webclient is in place and they are being disposed off after the usage. This should help manage memory in a better way after utilization of WebClient.

4.11. Potential memory overflow

Risk Level: Middle

Infinite recursion is possible in some cases in following code that may lead the recursion to continue until the stack overflows and the program crashes.
As a rule of thumb it is always suggested to provide a method to break out of recursion.



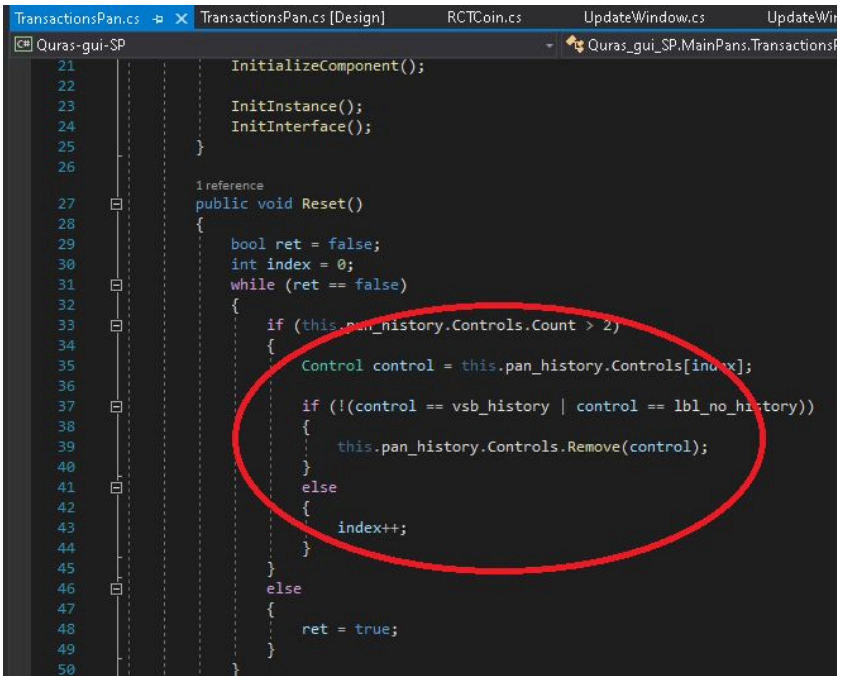
```
                    state = value;
                    ITrackable<RCTCoinReference> _this = this;
                    if (_this.TrackState == TrackState.None)
                        _this.TrackState = TrackState.Changed;
                }
            }
        }

        0 references
        public void SetTrackState(TrackState state)
        {
            ITrackable<RCTCoinReference> _this = this;
            _this.TrackState = state;
        }

        99+ references
        public bool Equals(RCTCoin other)
        {
            if (ReferenceEquals(this, other)) return true;
            if (ReferenceEquals(null, other)) return false;
            return Reference.Equals(other.Reference);
        }

        -references
        public override bool Equals(object obj)
        {
            return Equals(obj as JSCoin);
        }

        -references
        public override int GetHashCode()
        {
            return Reference.GetHashCode();
        }
    }
}
```

Fixed
TECHFUND verifies that the code has been fixed and no longer is an issue for overflow to happen.

4.12. Wrong removal of Pan History

Risk Level: Middle

It looks like a faulty logic to remove from pan_history, potentially this will lead to logic errors and might not work as intended. It is suggested to verify the bitwise operator of the if syntax.



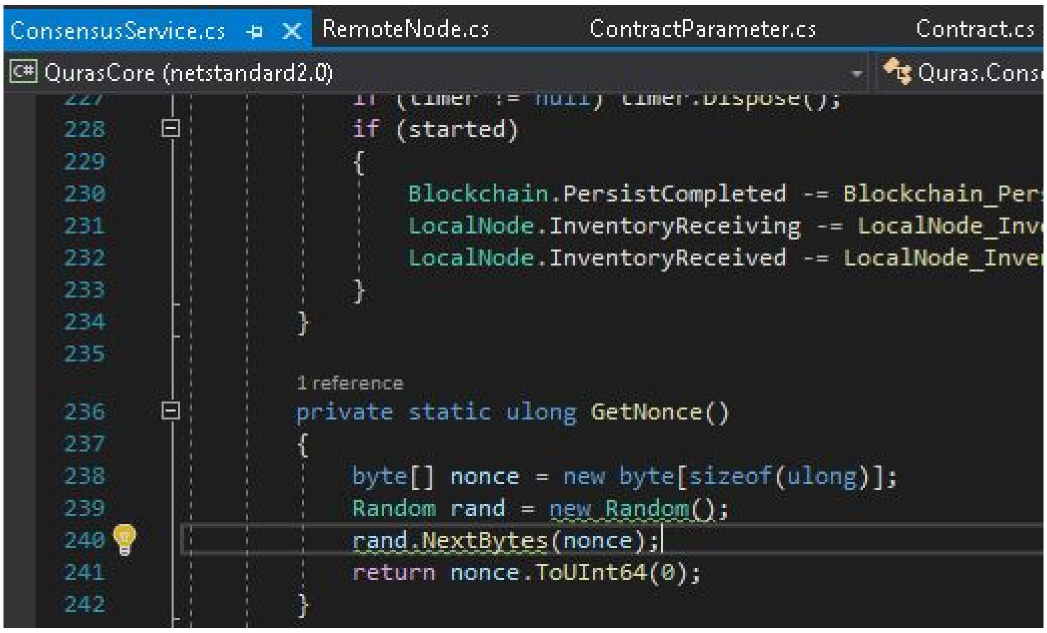
```
            InitializeComponent();

            InitInstance();
            InitInterface();
        }

        1 reference
        public void Reset()
        {
            bool ret = false;
            int index = 0;
            while (ret == false)
            {
                if (this.pan_history.Controls.Count > 2)
                {
                    Control control = this.pan_history.Controls[index];

                    if (!(control == vsb_history | control == lbl_no_history))
                    {
                        this.pan_history.Controls.Remove(control);
                    }
                    else
                    {
                        index++;
                    }
                }
                else
                {
                    ret = true;
                }
            }
```

Fixed TECHFUND verifies that the logic is now correct for pan_history removal.

4.13. Dependence on weak Randomness

Risk Level: Note

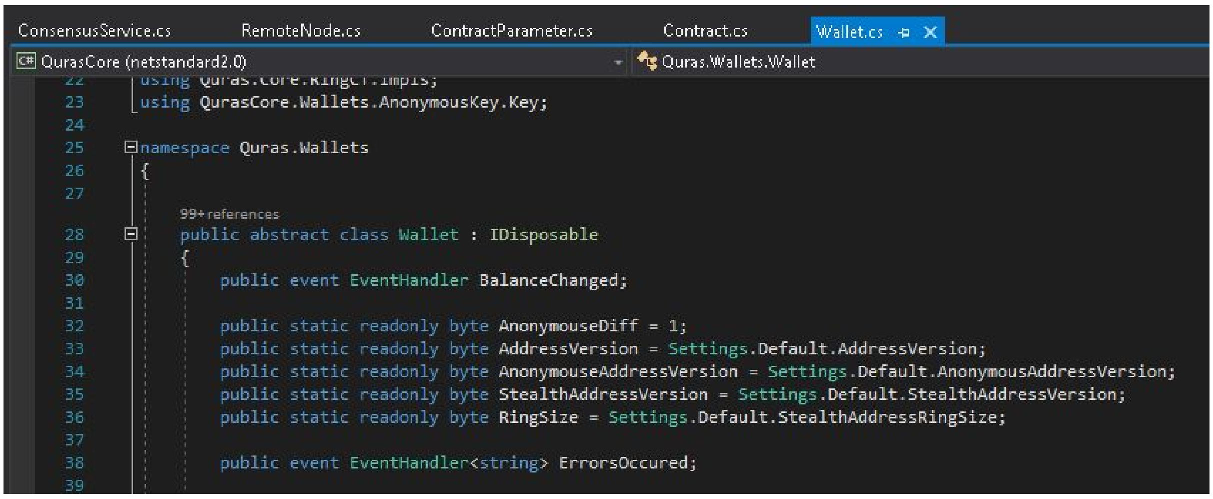
QURAS utilizes weak randomness at a lot of places and is strongly suggested against it. The algorithm used by the implementation of System.Random is weak because random numbers generated can be predicted.



ConsensusService.cs RemoteNode.cs ContractParameter.cs Contract.cs

QurasCore (netstandard2.0) Quras.Cons

```
            if (timer != null) timer.Dispose();
            if (started)
            {
                Blockchain.PersistCompleted -= Blockchain_Per
                LocalNode.InventoryReceiving -= LocalNode_Inv
                LocalNode.InventoryReceived -= LocalNode_Inve
            }
        }

        1 reference
        private static ulong GetNonce()
        {
            byte[] nonce = new byte[sizeof(ulong)];
            Random rand = new Random();
            rand.NextBytes(nonce);
            return nonce.ToUInt64(0);
        }
```



ConsensusService.cs RemoteNode.cs ContractParameter.cs Contract.cs Wallet.cs

QurasCore (netstandard2.0) Quras.Wallets.Wallet

```
using Quras.Core.RingCT.Impls;
using QurasCore.Wallets.AnonymousKey.Key;

namespace Quras.Wallets
{

    99+ references
    public abstract class Wallet : IDisposable
    {
        public event EventHandler BalanceChanged;

        public static readonly byte AnonymouseDiff = 1;
        public static readonly byte AddressVersion = Settings.Default.AddressVersion;
        public static readonly byte AnonymouseAddressVersion = Settings.Default.AnonymousAddressVersion;
        public static readonly byte StealthAddressVersion = Settings.Default.StealthAddressVersion;
        public static readonly byte RingSize = Settings.Default.StealthAddressRingSize;

        public event EventHandler<string> ErrorsOccured;
```

Quras Network > Local Node

Quras Network > Remote Node

Quras Core > Ring CT

Fixed

TECHFUND verifies that the code for RingCT / Local Node and Remote Node now handles random generators properly and dependence on weak randomness has been removed.

Apart from above we found that code practices can be improved and optimized overall, and can be simplified a lot, but they don’t portray any security threat, but QURAS team might have to work towards making code more cleaner. Some areas of improvement might be avoiding big methods and many parameters, avoiding types that are too long, removing dead code and dead types, handling errors in a better way to output relevant error code and so on.