```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace Tscan
{
    static class Tscan
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        ///
        public static Scanner Scan;
        [MTAThread]
        static void Main(String[] args)
        {
            System.Threading.ThreadPool.SetMinThreads(10, 10);
            System.Threading.ThreadPool.SetMaxThreads(Environment.ProcessorCount * 20, 10);
            if (args.Count() == 0)
            {
                Application.EnableVisualStyles();
                Application.SetCompatibleTextRenderingDefault(false);
                Application.Run(new ScanType());
            }
            else
            {
                Scan = new Scanner();
                if (args.Count() > 0)
                {
                    foreach (String arg in args)
                    {
                        if (arg.StartsWith("//Type", StringComparison.CurrentCultureIgnoreCase))
                        {
                            if (!int.TryParse(arg.Split(":".ToCharArray())[1], out Scan.IntScanType))
                            {
                                if (arg.Split(":".ToCharArray())[1].Equals("ThisMachine",
                                    StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 1;
                                else if (arg.Split(":".ToCharArray())[1].Equals("ServerList",
                                    StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 2;
                                else if (arg.Split(":".ToCharArray())[1].Equals("Subnet",
                                    StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 3;
                                else if (arg.Split(":".ToCharArray())[1].Equals("ActiveDirectory",
                                    StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 4;
                            }
                        }
                        else if (arg.StartsWith("//File", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.ServerListFilename = arg.Split(":".ToCharArray())[1];
                        }
                        else if (arg.StartsWith("//Password", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.Password = arg.Split(":".ToCharArray())[1];
                        }
                        else if (arg.StartsWith("//SearchTerms", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.SearchTerm = arg.Split(":".ToCharArray())[1];
                        }
                        else if (arg.StartsWith("//Script", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.RemoteExec.RemoteExecScript = arg.Split(":".ToCharArray())[1];
                        }
                        else if (arg.StartsWith("//WMIObjects", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.SearchObjects = arg.Split(":".ToCharArray())[1];
                        }
                        else if (arg.StartsWith("//ADOnly", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.ADOnly = true; ;
                        }
                        else if (arg.StartsWith("//Internet", StringComparison.CurrentCultureIgnoreCase))
                        {
                            Scan.ScanInternet = true; ;
                        }
                    }
                    Scan.ScanNetQueueWorkItem("");
                }
            }
        }
    }

    public class Scanner
```

```
    {
        [DllImport("IPHLPAPI.DLL", ExactSpelling = true)]
        public static extern int SendARP(uint DestIP, uint SrcIP, byte[] pMacAddr, ref uint PhyAddrLen);

        public int IntScanType;
        public Boolean ADOnly;
        public Boolean ScanInternet;
        Int32 IntDone;
        Int32 IntSQL;
        Int32 IntPort;
        Int32 IntServer;
        public String Password;
        public String SearchTerm;
        public String SearchObjects;
        public String MACLookupURI;
        public String[] WMIPasswords;
        public String[] WMIUsernames;
        public String[] XMLElements;
        public String ServerListFilename;
        //public System.Collections.Specialized.StringDictionary ServerList;
        public System.Collections.Concurrent.ConcurrentDictionary<String, String> ServerList;
        public ScannerActiveDirectory ScanAD;
        public ScannerRemoteExec RemoteExec;
        public Progress ProgressForm;
        /// <summary>
        /// Constructor for the main object
        /// </summary>
        ///
        public Scanner()
        {
            IntScanType = 0;
            ADOnly = false;
            IntDone = 0;
            IntSQL = 0;
            IntPort = 0;
            ServerListFilename = "";
            ServerList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
            ScanAD = new ScannerActiveDirectory();
            RemoteExec = new ScannerRemoteExec();
            String[] WMIPasswordsTemp = {Password, "password", "123456", ""};
            WMIPasswords = WMIPasswordsTemp;
            //SplashData 4+2% limited to avoid account lockout
            String[] WMIUsernamesTemp = {Environment.UserName, "administrator",
                "Administrator", "user1", "admin", "demo", "db2admin", "Admin", "sql"};
            WMIUsernames = WMIUsernamesTemp;
            //rapid7 several are 1/10k
        }
        /// <summary>
        /// This writes a file to disk
        /// </summary>
        ///
        public void WriteToDisk(String Title, String Table)
        {
            Boolean Caught = true;
            for (Int16 i = 0; i < 3 && Caught; i++)
            {
                if (i > 0) System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
                Caught = false;
                try
                {
                    System.IO.File.WriteAllText(System.IO.Path.Combine(
                        Environment.CurrentDirectory, Title), Table);
                }
                catch (System.IO.IOException e)
                {
                    MessageBox.Show(e.Message);
                    Caught = true;
                }
            }
        }
        /// <summary>
        /// The setup for the threadpool and a new thread for the scanner
        /// </summary>
        ///
        public void ScanNetQueueWorkItem(String ProgressLabel)
        {
            //MACLookup("");
            System.Threading.ThreadPool.QueueUserWorkItem(
                new System.Threading.WaitCallback(ScanNetWorkItem), ProgressLabel);
        }
        /// <summary>
        /// This adjusts the size of the thread pool to fit the processor and memory. Deprecated.
        /// </summary>
```

```
        ///
        public Boolean AreResourcesAvailable()
        {
            int[] MaxThreads = { 0, 0 };
            System.Threading.ThreadPool.GetMaxThreads(out MaxThreads[0], out MaxThreads[1]);
            System.Diagnostics.PerformanceCounter CPUCounter;
            System.Diagnostics.PerformanceCounter RamCounter;
            CPUCounter = new System.Diagnostics.PerformanceCounter("Processor", "% Processor Time", "_Total");
            RamCounter = new System.Diagnostics.PerformanceCounter("Memory", "Available MBytes");
            float CPUCountAvg = 0;
            for (int i = 0; i < 10; i++)
            {
                CPUCountAvg += CPUCounter.NextValue();
                System.Threading.Thread.Sleep(TimeSpan.FromSeconds(1));
                //10 seconds per 144 threads on 1 hour threads is 52k threads per hour
                //This reduces 1 second spikes associated with application launches
            }
            if (CPUCountAvg / 10 > 80 && MaxThreads[0] > Environment.ProcessorCount * 20) //20% free CPU
            {
                MaxThreads[0] -= Environment.ProcessorCount * 10;
                System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
                return false;
            }
            //else if (RamCounter.NextValue() < 200 && MaxThreads[0] > Environment.ProcessorCount * 20) //200MB
            free
            //{
            //    MaxThreads[0] -= Environment.ProcessorCount * 10;
            //    System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
            //    return false;
            //} //10MB per thread
            else if (MaxThreads[0] < Environment.ProcessorCount * 200) //Maxes out in 10 seconds * 20 steps is
            3 min
            {
                MaxThreads[0] += Environment.ProcessorCount * 10;
                System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
                return true;
            }
            else
            {
                return false;
            }
        }
        /// <summary>
        /// This method updates progress
        /// </summary>
        ///
        public void CountProgress()
        {
            IntPort = 0;
            IntDone = 0;
            IntSQL = 0;
            foreach (String File in System.IO.Directory.GetFiles(Environment.CurrentDirectory))
            {
                if (File.EndsWith("Ping.csv", StringComparison.CurrentCultureIgnoreCase))
                {
                    IntPort++;
                }
                else if (File.EndsWith("Done.txt", StringComparison.CurrentCultureIgnoreCase))
                {
                    IntDone++;
                }
                else if (File.EndsWith("SqlServiceAdvancedProperty.csv",
                StringComparison.CurrentCultureIgnoreCase))
                {
                    IntSQL++;
                }
            }
        }
        /// <summary>
        /// This method updates progress
        /// </summary>
        ///
        public void UpdateProgress(String CurrentComputer)
        {
            //CountProgress();
            int[] MaxThreads = { 0, 0 };
            System.Threading.ThreadPool.GetMaxThreads(out MaxThreads[0], out MaxThreads[1]);
            int[] AvailableThreads = { 0, 0 };
            System.Threading.ThreadPool.GetAvailableThreads(out AvailableThreads[0], out AvailableThreads[1]);
            String UpdateProgress = "Computers completed: " + IntDone + Environment.NewLine +
                "SQL computers completed: " + IntSQL + Environment.NewLine +
                "Servers completed: " + IntServer + Environment.NewLine +
                "Port scans completed: " + IntPort + Environment.NewLine +
```

```
                "Currently working on: " + CurrentComputer + Environment.NewLine +
                "Total computers: " + ServerList.Count + Environment.NewLine +
                (IntScanType == 4 ? "Total domains: " + ScanAD.DomainList.Count + Environment.NewLine : "") +
                "Concurrent work items: " + (MaxThreads[0] - AvailableThreads[0]) + "/" + MaxThreads[0];
            Tscan.Scan.ProgressForm.ProgressLabel.Invoke((MethodInvoker)(() =>
                Tscan.Scan.ProgressForm.ProgressLabel.Text = UpdateProgress));
        }
        /// <summary>
        /// This method requests threaded scans for globs of 10 servers each
        /// </summary>
        ///
        public void ScanNetWorkItem(Object ProgressLabel)
        {
            String[] SubsetServerList = new String[10];
            //test[1] = "str";
            SubsetServerList[0] = "Empty";
            RemoteExec.CompileService();
            BuildServerList();
            String[] ServerListArray;
            int[] AvailableThreads = { 0, 0 };
            for (int k = 0; k < 15; k++)
            {
                ServerListArray = new String[ServerList.Count];
                ServerList.Keys.CopyTo(ServerListArray, 0);
                for (Int32 i = 0; i <= ServerListArray.Count(); i += 10)
                {
                    SubsetServerList = new String[10];
                    for (int j = 0; j < 10 && i + j < ServerListArray.Count(); j++)
                    { SubsetServerList[j] = ServerListArray[i + j]; }
                    System.Threading.ThreadPool.QueueUserWorkItem(
                        new System.Threading.WaitCallback(Tscan.Scan.ScanTenServers), SubsetServerList);
                    if(i % (Environment.ProcessorCount * 20) == 0)AreResourcesAvailable();
                    UpdateProgress(SubsetServerList[0]);
                    System.Threading.ThreadPool.GetAvailableThreads(
                        out AvailableThreads[0], out AvailableThreads[1]);
                    while (AvailableThreads[0] == 0)
                    {
                        System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
                        AreResourcesAvailable();
                        UpdateProgress(SubsetServerList[0]);
                        System.Threading.ThreadPool.GetAvailableThreads(
                            out AvailableThreads[0], out AvailableThreads[1]);
                    }
                }
                for (Int32 i = 0; i < 60 * 8; i++)
                {
                    System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
                    AreResourcesAvailable();
                    UpdateProgress("8 Hour Wait. " + i + "/480 " + k + "/15");
                    if (IntDone >= ServerList.Count * 0.998) Application.Exit();
                }
                IntPort = 0;
            }
            //return 0;
        }
        /// <summary>
        /// This scans ten servers
        /// </summary>
        ///
        public void ScanTenServers(Object SubsetServerList)
        {
            //String[] str =
            //String[] str = ((System.Collections.IEnumerable)SubsetServerList).Cast<object>()
            //    .Select(x => (x!=null?x.ToString():null)).ToArray();
            foreach (String Server in (System.Collections.IEnumerable)SubsetServerList)
            {
                ScanServer(Server);
            }
        }
        /// <summary>
        /// This tries to scan using HTTP Headers
        /// </summary>
        ///
        public Boolean HTTPHeaderScan(String Server)
        {
            String[] Ports = { "80", "443", "8080", "8008", "8443" };
            foreach (String Port in Ports)
            {
                String StringTable = "";
                String StringHeader = "";
                String StringRow = "";
                Boolean HeaderDone = false;
                System.Net.HttpWebRequest Req;
```

```
330                 if (Port.Contains("443") || Port.Equals("443", StringComparison.CurrentCultureIgnoreCase))
331                 {
332                     Req = (System.Net.HttpWebRequest)System.Net.HttpWebRequest.Create(
333                     "https://" + Server + ":" + Port);
334                 }
335                 else
336                 {
337                     Req = (System.Net.HttpWebRequest)System.Net.HttpWebRequest.Create(
338                     "http://" + Server + ":" + Port);
339                 }
340                 System.Net.HttpWebResponse Resp;
341                 try
342                 {
343                     Resp = (System.Net.HttpWebResponse)Req.GetResponse();
344                     StringRow = "\"" + Server + "\",";
345                     StringHeader = "Computer,";
346                     foreach (String Key in Resp.Headers.AllKeys)
347                     {
348                         try
349                         {
350                             StringRow += "\"" + Resp.Headers[Key] + "\",";
351                         }
352                         catch
353                         {
354                             StringRow += "\"\",";
355                         }
356                         StringHeader += "\"" + Key + "\",";
357                     }
358                     if (!HeaderDone) StringTable = StringHeader + Environment.NewLine;
359                     HeaderDone = true;
360                     StringTable += StringRow + Environment.NewLine;
361                     Resp.Close();
362                     WriteToDisk(Server + "_" + Port + "_Header.csv", StringTable);
363                 }
364                 catch //(System.Net.WebException e)
365                 {
366 
367                 }
368             }
369             return true;
370         }
371         /// <summary>
372         /// This tries to scan using SMB Headers
373         /// </summary>
374         ///
375         public Boolean SMBHeaderScan(String Server)
376         {
377             String StringTable = "";
378             String StringHeader = "";
379             String StringRow = "";
380             Boolean HeaderDone = false;
381             System.Net.FileWebRequest Req =
382                 (System.Net.FileWebRequest)System.Net.FileWebRequest.Create("//" + Server +
                    "/admin$/notepad.exe");
383             System.Net.FileWebResponse Resp;
384             try
385             {
386                 Resp = (System.Net.FileWebResponse)Req.GetResponse();
387                 StringRow = "\"" + Server + "\",";
388                 StringHeader = "Computer,";
389                 foreach (String Key in Resp.Headers.AllKeys)
390                 {
391                     try
392                     {
393                         StringRow += "\"" + Resp.Headers[Key] + "\",";
394                     }
395                     catch
396                     {
397                         StringRow += "\"\",";
398                     }
399                     StringHeader += "\"" + Key + "\",";
400                 }
401                 if (!HeaderDone) StringTable = StringHeader + Environment.NewLine;
402                 HeaderDone = true;
403                 StringTable += StringRow + Environment.NewLine;
404                 Resp.Close();
405             }
406             catch (System.Net.WebException)
407             {
408                 return true;
409             }
410             WriteToDisk(Server + "_SMBHeader.csv", StringTable);
411             return true;
```

```
        }
        /// <summary>
        /// This icmp pings, syn pings 20 ports, and arp pings to get the MAC vendor for ping.csv
        /// </summary>
        ///
        public Boolean PortScan(String Server)
        {
            //ICMP - Broadcast, IP, Mac
            //Syn - Port, State, Service, Version
            //Nbtstat - Name, User, Mac
            //SMB - OS
            String Header = "";
            String Values = "";
            String Table = "";
            Byte[] MacAddressBytes;
            String Mac;
            Int16[] ListOfPorts = {53, //DNS
                                   80, //HTTP
                                   135, //WMI
                                   137,138,139, //NetBIOS
                                   389, //LDAP
                                   443, //HTTPS
                                   445, //SMB
                                   1433,1434, //SQL
                                   5060,5061, //SIP
                                   5431, //UPNP
                                   8008,8080,8443, //HTTP
                                   9100,9220 //PCL
                                   };
            Header = "Computer,";
            Values = "\"" + Server + "\",";
            try
            {
                System.Net.NetworkInformation.PingReply PR = new
                System.Net.NetworkInformation.Ping().Send(Server);
                Values += "\"" + PR.RoundtripTime + "\",";
                Header += "\"" + "PingTime" + "\",";
                Values += "\"" + PR.Status + "\",";
                Header += "\"" + "PingStatus" + "\",";
            }
            catch (System.Net.NetworkInformation.PingException e)
            {
                Values += "\"" + 0 + "\",";
                Header += "\"" + "PingTime" + "\",";
                Values += "\"" + e.Message + "\",";
                Header += "\"" + "PingStatus" + "\",";
            }
            foreach (Int16 IntPort in ListOfPorts)
            {
                Values += "\"" + SinglePort(Server, IntPort) + "\",";
                Header += "\"" + IntPort + "\",";
            }
            Header += "\"Mac\",\"Details\",";
            System.Net.IPAddress ipAddress = System.Net.IPAddress.Parse("0.0.0.0");
            Boolean Fail = false;
            try
            {
                Int16 Skip = 0x0;
                if (Server.Split(".".ToCharArray()).Count() == 4 &&
                Int16.TryParse(Server.Split(".".ToCharArray())[0], out Skip) &&
                Int16.TryParse(Server.Split(".".ToCharArray())[1], out Skip) &&
                Int16.TryParse(Server.Split(".".ToCharArray())[2], out Skip) &&
                Int16.TryParse(Server.Split(".".ToCharArray())[3], out Skip))
                {
                    //if server is IP
                    ipAddress = System.Net.IPAddress.Parse(Server);
                }
                else
                {
                    //if server is hostname
                    ipAddress = System.Net.Dns.GetHostEntry(Server).AddressList[0];
                }
            }
            catch
            {
                Fail = true;
            }
            if (!Fail)
            {
                uint UintAddress = BitConverter.ToUInt32(ipAddress.GetAddressBytes(), 0);
                uint MacAddressByteLength = 6;
                MacAddressBytes = new Byte[MacAddressByteLength];
                int SendARPSuccess = SendARP(UintAddress, 0, MacAddressBytes, ref MacAddressByteLength);
```

```
                if (SendARPSuccess == 0)
                {
                    Values += "\"";
                    Mac = "";
                    for (int i = 0; i < MacAddressByteLength; i++)
                    {
                        if (!String.IsNullOrEmpty(MacAddressBytes[i].ToString("x2")))
                            Mac += MacAddressBytes[i].ToString("x2");
                        else Mac += "--";
                        if (i < MacAddressByteLength - 1) Mac += ":";
                    }
                    Values += Mac;
                    Values += "\",";
                    Values += "\"";
                    Values += MACLookup(Mac);
                    Values += "\",";
                }
                else
                {
                    Values += "\"ARP Fail\",\"\",";
                }
            }
            else
            {
                Values += "\"DNS Fail\",\"\",";
            }
            Table = Header + Environment.NewLine + Values;
            WriteToDisk(Server + "_Ping.csv", Table);
            return true;
        }
        /// <summary>
        /// This looks up a MAC to it's vendor only. Runs 200 times an hour.
        /// </summary>
        ///
        public String MACLookup(String MAC)
        {
            //https://api.macvendors.co/'MAC'/xml
            //https://api.macvendors.com/'MAC'
            System.Net.HttpWebRequest Req =
                (System.Net.HttpWebRequest)System.Net.HttpWebRequest.Create(MACLookupURI.Replace("'MAC'", MAC));
            System.Net.HttpWebResponse Resp;
            String StringResponse = "";
            try
            {
                Resp = (System.Net.HttpWebResponse)Req.GetResponse();
                System.IO.Stream Streamer = Resp.GetResponseStream();
                System.IO.StreamReader StreamRead = new System.IO.StreamReader(Streamer);
                StringResponse = StreamRead.ReadToEnd();
                StreamRead.Close();
                Resp.Close();
            }
            catch(System.Net.WebException e)
            {
                StringResponse = e.Message;
            }
//            StringResponse = @"<result>
//<company>Apple, Inc.</company>
//<mac_prefix>08:74:02</mac_prefix>
//<address>1 Infinite Loop,Cupertino CA 95014,US</address>
//<start_hex>087402000000</start_hex>
//<end_hex>087402FFFFFF</end_hex>
//<country>US</country>
//<type>MA-L</type>
//</result>";
            if (StringResponse.Contains("json"))
            {
                //system.runtime.serialization json .net 4.0
                //System.Runtime.Serialization.json
            }
            else if (StringResponse.Contains("</"))
            {

                //system.xml xml
                System.Xml.XmlDocument Doc = new System.Xml.XmlDocument();
                Doc.LoadXml(StringResponse);
                //XMLElements
                System.Xml.XmlElement OneElement = null;
                foreach (String ElementName in XMLElements)
                {
                    if(OneElement == null)
                    {
                        OneElement = Doc[ElementName];
                    }
```

```
                else
                {
                    OneElement = OneElement[ElementName];
                }
            }
            StringResponse = ScrubString(OneElement.InnerText.ToString());
        }
        else
        {
            StringResponse = ScrubString(StringResponse);
        }
        return StringResponse;
    }
    /// <summary>
    /// This scrubs a string of comma, quote, space
    /// </summary>
    ///
    public String ScrubString(String String)
    {
        return String.Replace(Environment.NewLine, " ").Replace("\"", "").Replace(",", " ");
    }
    /// <summary>
    /// This pings a single port and determines if it's open
    /// </summary>
    ///
    public Boolean SinglePort(String Server, Int16 Port)
    {
        //Port, State, Service, Version - nmap
        //ACK,PSH,RST,SYN,FIN
        try
        {
            System.Net.Sockets.TcpClient Client = new System.Net.Sockets.TcpClient();
            Client.Connect(Server, Port);
            return Client.Connected;
        }
        catch (System.Net.Sockets.SocketException)
        {
            //MessageBox.Show(e.Message);//SocketException is common for offline machines
            return false;
        }
    }
    /// <summary>
    /// This sets up the WMI scope
    /// </summary>
    ///
    public System.Management.ManagementScope SetupScope(String Server, String Domain, String User, String
    Pass)
    {
        System.Management.ManagementScope Scope = new System.Management.ManagementScope();
        Scope = new System.Management.ManagementScope("\\\\" + Server + "\\root\\cimv2");
        Scope.Options.Authentication = System.Management.AuthenticationLevel.Packet;
        Scope.Options.EnablePrivileges = true;
        Scope.Options.Impersonation = System.Management.ImpersonationLevel.Impersonate;
        Scope.Options.Locale = "MS_409";
        Scope.Options.Timeout = TimeSpan.FromMinutes(10);
        if (User == "" && Pass == "")
        {
        }
        else if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase))
        {
            Scope.Options.Password = Pass;
            Scope.Options.Username = Domain + "\\" + User;
        }
        else
        {
            Scope.Options.Password = Pass;
            Scope.Options.Username = User;
            Scope.Options.Authority = Domain;
        }
        return Scope;
    }
    /// <summary>
    /// This determines if a username and password are valid
    /// </summary>
    ///
    public Int16 TestPassword(String Server, String Domain, String User, String Pass)
    {
        String Table = "";
        String OSQuery = "Select * from win32_operatingsystem";
        System.Management.ManagementScope Scope = SetupScope(Server, Domain, User, Pass);
        try
        {
            Scope.Connect();
```

```
            }
            catch (Exception f)
            {
                if (f.Message == "The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)") return 2;
                return 0;
            }
            System.Management.ObjectQuery Query = new System.Management.ObjectQuery(OSQuery);
            System.Management.ManagementObjectSearcher Searcher =
                new System.Management.ManagementObjectSearcher(Scope, Query);
            try
            {
                System.Management.ManagementObjectCollection Result = Searcher.Get();
                if (Result.Count == 0) return 0;
            }
            catch (Exception f)
            {
                if (f.Message == "The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)") return 2;
                return 0;
            }
            if (!String.IsNullOrEmpty(User))
            {
                Table = "Computer,\"Domain\",\"Username\",\"Password\"" + Environment.NewLine +
                    "\"" + Server + "\",\"" + Domain + "\",\"" + User + "\",\"" + Pass + "\"";
                WriteToDisk(Server + "_Password.csv", Table);
            }
            return 1;
        }
        /// <summary>
        /// This collects a csv from a single WMI Object formatted as a table
        /// </summary>
        ///
        public Boolean SingleWMITable(String Server, String Domain, String Object, String User, String Pass,
        String Namespace)
        {
            String OSQuery = "Select * from " + Object;
            System.Management.ManagementScope Scope = SetupScope(Server, Domain, User, Pass);
            Scope.Path = new System.Management.ManagementPath("\\\\" + Server + "\\" + Namespace);
            try
            {
                Scope.Connect();
            }
            catch
            {
                return false;
            }
            System.Management.ObjectQuery Query = new System.Management.ObjectQuery(OSQuery);
            System.Management.ManagementObjectSearcher Searcher =
                new System.Management.ManagementObjectSearcher(Scope, Query);
            String Names = "";
            String Values = "";
            String Table = "";
            Boolean HeaderDone = false;
            Boolean RemoteExecDone = false;
            System.Management.ManagementObjectCollection Result;
            try
            {
                Result = Searcher.Get();
                if (Result.Count == 0) return false;
            }
            catch
            {
                return false;
            }
            try
            {
                foreach (System.Management.ManagementBaseObject Row in Result)
                {
                    Names = "Computer,";
                    Values = "\"" + Server + "\",";
                    foreach (System.Management.PropertyData Cell in Row.Properties)
                    {
                        Names += "\"" + Cell.Name + "\",";
                        if (Cell.Value != null)
                        {
                            String CellValue = "";
                            if (Cell.Value.ToString().Equals("System.String[]") ||
                                Cell.Value.ToString().Equals("System.UInt16[]"))
                            {
                                String[] CellArray = ((System.Collections.IEnumerable)Cell.Value).Cast<object>()
                                    .Select(x => (x != null ? x.ToString() : null)).ToArray();
                                if (CellArray.Count() < 10)
                                    foreach (String CellArrayValue in CellArray)
                                        CellValue += CellArrayValue + " ";
```

```
                                else CellValue = Cell.Value.ToString();
                            }
                            else CellValue = Cell.Value.ToString();
                            Values += "\"" + ScrubString(CellValue) + "\",";
                            if (SearchObjects.ToLower().Contains(Object.ToLower()))
                            {
                                foreach (String FindWord in SearchTerm.Split(",".ToCharArray()))
                                {
                                    if (FindWord.Length > 2
                                        && ScrubString(CellValue).ToLower().Contains(FindWord.ToLower())
                                        && !RemoteExecDone)
                                        RemoteExecDone = RemoteExec.RemoteExec(Server, Domain, User, Pass);
                                }
                                if (Cell.Name.Equals("Name", StringComparison.CurrentCultureIgnoreCase)
                                    && Object.Equals("win32_operatingsystem",
                                    StringComparison.CurrentCultureIgnoreCase)
                                    && ScrubString(CellValue).ToLower().Contains("windows server"))
                                    IntServer++;
                            }
                        }
                        else
                        {
                            Values += "\"\",";
                        }
                    }
                    if (IntScanType == 3 &&
                            Object.Equals("win32_useraccount", StringComparison.CurrentCultureIgnoreCase) &&
                            Row.GetPropertyValue("LocalAccount").ToString().Equals("False",
                            StringComparison.CurrentCultureIgnoreCase) &&
                            Row.GetPropertyValue("Disabled").ToString().Equals("False",
                            StringComparison.CurrentCultureIgnoreCase) &&
                            !Tscan.Scan.ScanAD.DomainAdminList.ContainsKey(Row.GetPropertyValue("Domain") +
                            "\\" +
                            Row.GetPropertyValue("Name"))
                            )
                    {
                        foreach (String WMIPass in Tscan.Scan.WMIPasswords)
                        {
                            if (TestPassword(Server,
                                Row.GetPropertyValue("Domain").ToString(),
                                Row.GetPropertyValue("Name").ToString(),
                                WMIPass).Equals(1))
                            {
                                Tscan.Scan.ScanAD.DomainAdminList.TryAdd(Row.GetPropertyValue("Domain") + "\\" +
                                    Row.GetPropertyValue("Name"), WMIPass);
                                MessageBox.Show("Found password for " + Row.GetPropertyValue("Name"));
                                break;
                            }
                        }
                        if (!Tscan.Scan.ScanAD.DomainAdminList.ContainsKey(Row.GetPropertyValue("Domain") +
                            "\\" +
                            Row.GetPropertyValue("Name")))
                            Tscan.Scan.ScanAD.DomainAdminList.TryAdd(Row.GetPropertyValue("Domain") + "\\" +
                                Row.GetPropertyValue("Name"), "fail");
                    }
                    if (!HeaderDone) Table += Names + Environment.NewLine;
                    Table += Values + Environment.NewLine;
                    HeaderDone = true;
                }
            }
            catch(System.Runtime.InteropServices.COMException e)
            {
                MessageBox.Show(Server + " " + Object + " " + e.Message);
            }
            WriteToDisk(Server + "_" + Object + ".csv", Table);
            if (String.IsNullOrEmpty(Table)) return false;
            else return true;
        }
        /// <summary>
        /// This scans a single server
        /// </summary>
        ///
        public void ScanServer(String Server)
        {
            Boolean Success = true;
            Int16 Skip = 0x0;
            String UserSuccess = "";
            String PassSuccess = "";
            String DomainSuccess = "";
            String ServerOriginal = Server;
            //the next line may face problems with dhcp on Class C networks
            if (String.IsNullOrEmpty(Server)) return;
            if (ServerList[Server].Equals("Done", StringComparison.CurrentCultureIgnoreCase) &&
```

```
                    IntScanType != 3 && !(Server.Split(".".ToCharArray()).Count() == 4 &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[0], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[1], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[2], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[3], out Skip))) return;
                String ServerResolution = Resolve(Server);
                if (String.IsNullOrEmpty(ServerResolution)) return;
                Server = ServerResolution;
                try
                {
                    if (System.IO.File.Exists(System.IO.Path.Combine(
                        Environment.CurrentDirectory, Server + "_" + "Done" + ".txt")))
                    {
                        ServerList[ServerOriginal] = "Done";
                        return;
                    }
                }
                catch
                {
                    return;
                }
                //PortScan(Server);
                if (IntScanType == 3 &&
                    Server.Split(".".ToCharArray()).Count() == 4 &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[0], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[1], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[2], out Skip) &&
                    Int16.TryParse(Server.Split(".".ToCharArray())[3], out Skip) &&
                    !new System.Net.NetworkInformation.Ping().Send(Server).Status.Equals("Success") &&
                    TestPassword(Server, "", "", "").Equals(2))
                {
                    //if (PortScan(Server)) IntPort++;
                    //skip if scanning subnet, resolve failed, ping failed, and wmi port is closed or firewalled
                    return;
                }
                if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
                {
                    UserSuccess = "";
                    PassSuccess = "";
                }
                else if (TestPassword(Server, "", "", "").Equals(1))
                {
                    UserSuccess = "";
                    PassSuccess = "";
                }
                else if (TestPassword(Server, Server, Environment.UserName, Password).Equals(1))
                {
                    //likely to fail due to server as authority
                    DomainSuccess = Server;
                    UserSuccess = Environment.UserName;
                    PassSuccess = Password;
                }
                else
                {
                    foreach (String User in WMIUsernames)
                    {
                        foreach (String Pass in WMIPasswords)
                        {
                            if (TestPassword(Server, Server, User, Pass).Equals(1))
                            {
                                DomainSuccess = Server;
                                UserSuccess = User;
                                PassSuccess = Pass;
                                break;
                            }
                        }
                        if (!String.IsNullOrEmpty(UserSuccess)) break;
                    }
                    if(String.IsNullOrEmpty(UserSuccess))
                    {
                        foreach (String Key in ScanAD.DomainAdminList.Keys)
                        {
                            if (ScanAD.DomainAdminList[Key].Equals("fail") &&
                                TestPassword(Server,
                                Key.Split("\\".ToCharArray())[0],
                                Key.Split("\\".ToCharArray())[1],
                                ScanAD.DomainAdminList[Key]).Equals(1))
                            {
                                DomainSuccess = Key.Split("\\".ToCharArray())[0];
                                UserSuccess = Key.Split("\\".ToCharArray())[1];
                                PassSuccess = ScanAD.DomainAdminList[Key];
                                break;
                            }
```

```
                }
            }
        }
        if (!String.IsNullOrEmpty(UserSuccess))
        {
            WriteToDisk(Server + "_Password.csv",
                        "Computer,\"Domain\",\"User\",\"Pass\"" + Environment.NewLine +
                        "\"" + Server +
                        "\",\"" + DomainSuccess + "\",\"" + UserSuccess + "\",\"" + PassSuccess + "\"");
        }
        String[] WMIObjects = {"win32_product", "win32_quickfixengineering", //Software
                               "win32_service", "win32_operatingsystem", //Software
                               "win32_networkadapterconfiguration", "win32_processor", //HW
                               "win32_computersystem", "win32_systemenclosure", "win32_diskdrive", //HW
                               "win32_systemusers", "win32_useraccount", //user
                               "win32_groupuser", "win32_loggedonuser", //user
                               "win32_osrecoveryconfiguration", "win32_ntdomain", //sundry
                               "win32_perfformatteddata_perfnet_serverworkqueues", //perf
                               "win32_perfformatteddata_perfos_processor",  //perf
                               "win32_perfformatteddata_perfos_memory"}; //perf
        List<String> WMIObjectsList = WMIObjects.ToList();
        foreach (String WMIObject in SearchObjects.Split(",".ToCharArray()))
        {
            if (!WMIObjectsList.Contains(WMIObject.ToLower()))
            {
                WMIObjectsList.Add(WMIObject.ToLower());
            }
        }
        WMIObjects = WMIObjectsList.ToArray();
        foreach (String WMIObject in WMIObjects)
        {
            if (SingleWMITable(Server,
                DomainSuccess,
                WMIObject,
                UserSuccess,
                PassSuccess,
                "root\\cimv2") && Success)
                Success = true;
            else Success = false;
        }
        String[] SQLNamespaces = {"root\\Microsoft\\SqlServer\\ComputerManagement\\MSSQLSERVER",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement10\\MSSQLSERVER",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement12\\MSSQLSERVER",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement14\\MSSQLSERVER",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement10",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement12",
                                  "root\\Microsoft\\SqlServer\\ComputerManagement14"};
        Boolean BoolSQL = false;
        foreach (String SQLNamespace in SQLNamespaces)
        {
            if (SingleWMITable(Server,
                DomainSuccess,
                "SqlServiceAdvancedProperty",
                UserSuccess,
                PassSuccess,
                SQLNamespace))
                BoolSQL = true;
        }
        if (BoolSQL) IntSQL++;
        //5/6 complete
        String[] MSVMObjects = { "msvm_computersystem", "msvm_processor", "msvm_diskdrive",
                                 "msvm_summaryinformation", "msvm_guestnetworkadapterconfiguration",
                                 "msvm_syntheticethernetportsettingdata" };
        WMIObjectsList = MSVMObjects.ToList();
        foreach (String WMIObject in SearchObjects.Split(",".ToCharArray()))
        {
            if (!WMIObjectsList.Contains(WMIObject.ToLower()))
            {
                WMIObjectsList.Add(WMIObject.ToLower());
            }
        }
        MSVMObjects = WMIObjectsList.ToArray();
        foreach (String WMIObject in MSVMObjects)
        {
            SingleWMITable(Server,
                DomainSuccess,
                WMIObject,
                UserSuccess,
                PassSuccess,
                "root\\virtualization\\v2");
        }
        foreach (String WMIObject in MSVMObjects)
```

```
            {
                SingleWMITable(Server,
                    DomainSuccess,
                    WMIObject,
                    UserSuccess,
                    PassSuccess,
                    "root\\virtualization");
            }
            if (PortScan(Server) && Success) Success = true;
            if (SMBHeaderScan(Server) && Success) Success = true;
            if (HTTPHeaderScan(Server) && Success) Success = true;
            else Success = false;
            //I think port scan has only true return paths lol
            if (Success)
            {
                ServerList[ServerOriginal] = "Done";
                IntDone++;
                WriteToDisk(Server + "_Done.txt",
                            "Computer,Done" + Environment.NewLine + Server + "," + DateTime.Now.ToString());
            }
        }
        /// <summary>
        /// This resolves a single host. Sometimes this is an indicator of existance.
        /// </summary>
        ///
        public String Resolve(String Server)
        {
            String HostName = "";
            System.Net.IPHostEntry Host = new System.Net.IPHostEntry();
            try
            {
                Host = System.Net.Dns.GetHostEntry(Server);
                HostName = Host.HostName;
            }
            catch
            {
                return Server;
            }
            return HostName;
        }
        /// <summary>
        /// This builds a server list based on selections in ScanType.cs form.
        /// </summary>
        ///
        public void BuildServerList()
        {
            if (IntScanType == 1)
            {
                ServerList.TryAdd(Environment.MachineName, "");
            }
            else if (IntScanType == 2)
            {
                if (String.IsNullOrEmpty(ServerListFilename))
                    ServerListFilename =
                        System.IO.Path.Combine(Environment.CurrentDirectory, "Serverlist.txt");
                if (!System.IO.File.Exists(ServerListFilename))
                {
                    WriteToDisk(ServerListFilename, Environment.MachineName);
                }
                foreach (String Server in System.IO.File.ReadAllLines(ServerListFilename))
                {
                    ServerList.TryAdd(Server, "");
                }

            }
            else if (IntScanType == 3)
            {
                String Subnet = "";
                String MyIP = "";
                foreach (System.Net.NetworkInformation.NetworkInterface Interface in
                    System.Net.NetworkInformation.NetworkInterface.GetAllNetworkInterfaces())
                {
                    foreach (System.Net.NetworkInformation.UnicastIPAddressInformation Address in
                        Interface.GetIPProperties().UnicastAddresses)
                    {
                        if (Interface.OperationalStatus != System.Net.NetworkInformation.OperationalStatus.Down
                        &&
                            Address.IPv4Mask != null && "0.0.0.0" != Address.IPv4Mask.ToString() &&

                            !Address.Address.ToString().StartsWith("127",StringComparison.CurrentCultureIgnoreCas
                            e))
                        {
                            Subnet = Address.IPv4Mask.ToString();
```

```
                        MyIP = Address.Address.ToString();

                }
            }
        }
        String[] SubnetArray = Subnet.Split(".".ToCharArray());
        String[] MyIPArray = MyIP.Split(".".ToCharArray());
        if (SubnetArray[3] == "0" || SubnetArray[3] == "240")
        {
            for (int a = 1; a <= 255; a++)
            {
                for (int b = 0; b <= 255; b++)
                {
                    for (int c = 0; c <= 255; c++)
                    {
                        for (int d = 1; d <= 254; d++)
                        {
                            if (ScanInternet)
                            {
                                if (!(a == 10 ||
                                    (a == 172 && (b >= 16 && b <= 31)) ||
                                    (a == 192 && b == 168)))
                                {
                                    ServerList.TryAdd(a.ToString() + "." + b + "." + c + "." + d, "");
                                    if (c == 0 && d == 0) UpdateProgress(a.ToString() + "." + b + "." +
                                        c + "." + d);
                                }
                                //Internet, I recommend 300GB of memory for the serverlist
                                //Pioneer 199.0.0.0
                                //4b addresses in 10 hours
                            }
                            else if (SubnetArray[0] == "0")
                            {
                                ServerList.TryAdd(a.ToString() + "." + b + "." + c + "." + d, "");
                                if (c == 0 && d == 0) UpdateProgress(a.ToString() + "." + b + "." +
                                    c + "." + d);
                                //no gateway, 169.254.0.1 4b addresses in 10 hours
                            }
                            else if (SubnetArray[1] == "0" && SubnetArray[0] == "255")
                            {
                                ServerList.TryAdd(MyIPArray[0] + "." + b + "." + c + "." + d, "");
                                if (c == 0 && d == 0) UpdateProgress(MyIPArray[0] + "." + b + "." +
                                    c + "." + d);
                                //c 10.0.0.1 16m addresses in 2 minutes
                            }
                            else if (SubnetArray[2] == "0" && SubnetArray[1] == "255")
                            {
                                ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
                                    c + "." + d, "");
                                //b 172.16.0.1 65k addresses in seconds
                            }
                            else if (SubnetArray[3] == "0" && SubnetArray[2] == "255")
                            {
                                ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
                                    MyIPArray[2] + "." + d, "");
                                //a 192.168.1.1 256 addresses built in seconds
                            }
                            else if (SubnetArray[3] != "255" && SubnetArray[3] != "0")
                            {
                                if ((d & Int16.Parse(SubnetArray[3])) ==
                                    (Int16.Parse(MyIPArray[3]) & Int16.Parse(SubnetArray[3])))
                                {
                                    ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
                                        MyIPArray[2] + "." + d, "");
                                }
                            }
                            else if (SubnetArray[2] != "255" && SubnetArray[2] != "0")
                            {
                                if ((c & Int16.Parse(SubnetArray[2])) ==
                                    (Int16.Parse(MyIPArray[2]) & Int16.Parse(SubnetArray[2])))
                                {
                                    ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." + c + "."
                                    + d, "");
                                }
                            }
                            else if (SubnetArray[1] != "255" && SubnetArray[1] != "0")
                            {
                                if ((b & Int16.Parse(SubnetArray[1])) ==
                                    (Int16.Parse(MyIPArray[1]) & Int16.Parse(SubnetArray[1])))
                                {
                                    ServerList.TryAdd(MyIPArray[0] + "." + b + "." + c + "." + d, "");
                                }
                            }
```

```
                                    else ServerList.TryAdd(MyIP, "");
                                    if (SubnetArray[3] == "255" && !ScanInternet) break;
                                }
                                if (SubnetArray[2] == "255" && !ScanInternet) break;
                            }
                            if (SubnetArray[1] == "255" && !ScanInternet) break;
                        }
                        if (SubnetArray[0] == "255" && !ScanInternet) break;
                    }
                }
            }
            else if (IntScanType == 4)
            {
                Boolean ADAvailable = true;
                try
                {
                    //get user domain
                    System.DirectoryServices.ActiveDirectory.Domain.GetCurrentDomain();
                }
                catch
                {
                    //if user authority isn't a domain then disable AD scans
                    //this.ActiveDirectory.Enabled = false;
                    //this.ADOnly.Enabled = false;
                    ADAvailable = false;
                }
                if (ADAvailable)
                {
                    ScanAD.ScanAdmins();
                    ScanAD.ScanActiveDirectory();
                }
                else
                {
                    ScanAD.ScanWinsAndSql();
                    if (ADOnly)
                    {
                        System.IO.File.WriteAllText("Serverlist.txt", "");
                        foreach (String Key in ServerList.Keys)
                            System.IO.File.AppendAllText("Serverlist.txt", Key + Environment.NewLine);
                    }
                }
                if (ADOnly) Application.Exit();
            }
            System.IO.File.WriteAllText("Serverlist.txt", "");
            foreach (String Key in ServerList.Keys)
                System.IO.File.AppendAllText("Serverlist.txt", Key + Environment.NewLine);
        }
    }
    public class ScannerRemoteExec
    {
        [DllImport("advapi32.dll", SetLastError = true, CharSet = CharSet.Auto)]
        public static extern IntPtr CreateService(
            IntPtr hSCManager,
            string lpServiceName,
            string lpDisplayName,
            uint dwDesiredAccess,
            uint dwServiceType,
            uint dwStartType,
            uint dwErrorControl,
            string lpBinaryPathName,
            string lpLoadOrderGroup,
            string lpdwTagId,
            string lpDependencies,
            string lpServiceStartName,
            string lpPassword);
        [DllImport("advapi32.dll", EntryPoint = "OpenSCManagerW", ExactSpelling = true,
            CharSet = CharSet.Unicode, SetLastError = true)]
        public static extern IntPtr OpenSCManager(string machineName, string databaseName, uint dwAccess);
        [DllImport("advapi32.dll", SetLastError = true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        public static extern bool CloseServiceHandle(IntPtr hSCObject);
        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern bool CloseHandle(IntPtr hHandle);
        public String RemoteExecScript;
        public String PathToService;
        public String ServiceTextToCompile = @"
using System;
using System.Diagnostics;
using System.ServiceProcess;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace WindowsService
```

```
{
    class WindowsService : ServiceBase
    {
        /// <summary>
        /// Public Constructor for WindowsService.
        /// - Put all of your Initialization code here.
        /// </summary>
        public WindowsService()
        {
            this.ServiceName = ""My Windows Service"";
            this.EventLog.Log = ""Application"";

            // These Flags set whether or not to handle that specific
            //  type of event. Set to true if you need it, false otherwise.
            this.CanHandlePowerEvent = true;
            this.CanHandleSessionChangeEvent = true;
            this.CanPauseAndContinue = true;
            this.CanShutdown = true;
            this.CanStop = true;
        }

        /// <summary>
        /// The Main Thread: This is where your Service is Run.
        /// </summary>
        static void Main()
        {
            ServiceBase.Run(new WindowsService());
        }

        /// <summary>
        /// Dispose of objects that need it here.
        /// </summary>
        /// <param name=""disposing"">Whether
        ///    or not disposing is going on.</param>
        protected override void Dispose(bool disposing)
        {
            base.Dispose(disposing);
        }

        /// <summary>
        /// OnStart(): Put startup code here
        ///  - Start threads, get inital data, etc.
        /// </summary>
        /// <param name=""args""></param>
        protected override void OnStart(string[] args)
        {
            base.OnStart(args);
            //String GoTo = ""OnStart"";
            try
            {
                //GotTo = ""Begin Try"";
                String RemoteExecScript = ""RemoteExecScriptString"";
                String[] SplitRemoteExecScript = RemoteExecScript.Split("" "".ToCharArray(), 2);
                System.Security.SecureString SecPass = new System.Security.SecureString();
                foreach (char PassChar in ""PasswordString"") SecPass.AppendChar(PassChar);
                System.Diagnostics.ProcessStartInfo RemoteExecStartInfo =
                    new System.Diagnostics.ProcessStartInfo(SplitRemoteExecScript[0], SplitRemoteExecScript[1]);
                RemoteExecStartInfo.Verb = ""runas"";
                RemoteExecStartInfo.UseShellExecute = false;
                RemoteExecStartInfo.UserName = ""UserNameString"";
                RemoteExecStartInfo.Password = SecPass;
                RemoteExecStartInfo.Domain = ""DomainNameString"";
                RemoteExecStartInfo.LoadUserProfile = true;
                RemoteExecStartInfo.WorkingDirectory = Environment.CurrentDirectory;
                RemoteExecStartInfo.CreateNoWindow = true;
                RemoteExecStartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
                RemoteExecStartInfo.RedirectStandardOutput = true;
                RemoteExecStartInfo.RedirectStandardError = true;
                System.Diagnostics.Process RemoteExecProcess = new System.Diagnostics.Process();
                RemoteExecProcess = System.Diagnostics.Process.Start(RemoteExecStartInfo);
                System.IO.StreamReader ReaderOutput = RemoteExecProcess.StandardOutput;
                System.IO.StreamReader ReaderError = RemoteExecProcess.StandardError;
                String StandardOutput = """";
                //GotTo = ""Before While"";
                try
                {
                    while (!RemoteExecProcess.HasExited)
                    {
                        StandardOutput += ReaderOutput.ReadToEnd();
                        StandardOutput += ReaderError.ReadToEnd();
                        RemoteExecProcess.WaitForExit(TimeSpan.FromSeconds(1).Milliseconds);

                    }
```

```
                }
                catch(System.ComponentModel.Win32Exception) {}
                //GotTo = ""Before Write Errors"";
                System.IO.File.WriteAllText(
                    System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToString()
                    ,
                    ""RemoteExecOutput.txt""), StandardOutput);
                if(RemoteExecProcess.ExitCode != 0)
                    System.IO.File.WriteAllText(
                        System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToStri
                        ng(),
                        ""RemoteExecExitCode.txt""), RemoteExecProcess.ExitCode.ToString());
            }
            catch (Exception e)
            {
                System.Diagnostics.StackTrace Stack = new System.Diagnostics.StackTrace(e);
                System.Diagnostics.StackFrame Frame = Stack.GetFrame(Stack.FrameCount - 1);
                Int32 Line = Frame.GetFileLineNumber();
                System.IO.File.WriteAllText(
                    System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToString()
                    ,
                    ""RemoteExecExitCode.txt""), e.Message + "" "" + e.GetType().FullName +"" "" + Line);
            }
            System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
        }
    /// <summary>
    /// OnStop(): Put your stop code here
    /// - Stop threads, set final data, etc.
    /// </summary>
    protected override void OnStop()
        {
            base.OnStop();
        }

        /// <summary>
        /// OnPause: Put your pause code here
        /// - Pause working threads, etc.
        /// </summary>
        protected override void OnPause()
        {
            base.OnPause();
        }

        /// <summary>
        /// OnContinue(): Put your continue code here
        /// - Un-pause working threads, etc.
        /// </summary>
        protected override void OnContinue()
        {
            base.OnContinue();
        }

        /// <summary>
        /// OnShutdown(): Called when the System is shutting down
        /// - Put code here when you need special handling
        ///   of code that deals with a system shutdown, such
        ///   as saving special data before shutdown.
        /// </summary>
        protected override void OnShutdown()
        {
            base.OnShutdown();
        }

        /// <summary>
        /// OnCustomCommand(): If you need to send a command to your
        ///   service without the need for Remoting or Sockets, use
        ///   this method to do custom methods.
        /// </summary>
        /// <param name=""command"">Arbitrary Integer between 128 & 256</param>
        protected override void OnCustomCommand(int command)
        {
            //  A custom command can be sent to a service by using this method:
            //#  int command = 128; //Some Arbitrary number between 128 & 256
            //#  ServiceController sc = new ServiceController(""NameOfService"");
            //#  sc.ExecuteCommand(command);

            base.OnCustomCommand(command);
        }

```

```
        /// <summary>
        /// OnPowerEvent(): Useful for detecting power status changes,
        ///   such as going into Suspend mode or Low Battery for laptops.
        /// </summary>
        /// <param name=""powerStatus"">The Power Broadcast Status
        /// (BatteryLow, Suspend, etc.)</param>
        protected override bool OnPowerEvent(PowerBroadcastStatus powerStatus)
        {
            return base.OnPowerEvent(powerStatus);
        }

        /// <summary>
        /// OnSessionChange(): To handle a change event
        ///   from a Terminal Server session.
        ///   Useful if you need to determine
        ///   when a user logs in remotely or logs off,
        ///   or when someone logs into the console.
        /// </summary>
        /// <param name=""changeDescription"">The Session Change
        /// Event that occured.</param>
        protected override void OnSessionChange(
                  SessionChangeDescription changeDescription)
        {
            base.OnSessionChange(changeDescription);
        }
    }
}

";
        /// <summary>
        /// This allows you to run a program on a the local system
        /// </summary>
        ///
        private void LocalExec(String Server, String User, String Pass)
        {
            //getuserhandle other code paths so unecessary
            //    EnablePrivilege(SE_DEBUG_NAME)
            //    EnablePrivilege(SE_RESTORE_NAME);
            //    EnablePrivilege(SE_BACKUP_NAME);
            //
            //Not use system account or current user
            //    EnablePrivilege(SE_ASSIGNPRIMARYTOKEN_NAME);
            //    EnablePrivilege(SE_INCREASE_QUOTA_NAME);
            //    EnablePrivilege(SE_IMPERSONATE_NAME);
            //    ImpersonateLoggedOnUser(hUser);
            // .net 4.0 system.security se impersonation
            try
            {
                String[] SplitRemoteExecScript =
                    RemoteExecScript.Replace("\\\"", "\"").Replace("\\\\", "\\").Split(" ".ToCharArray(), 2);
                System.Security.SecureString SecPass = new System.Security.SecureString();
                foreach (char PassChar in Tscan.Scan.Password) SecPass.AppendChar(PassChar);
                System.Diagnostics.ProcessStartInfo LocalExecStartInfo =
                    new System.Diagnostics.ProcessStartInfo(SplitRemoteExecScript[0], SplitRemoteExecScript[1]);
                LocalExecStartInfo.Verb = "runas";
                LocalExecStartInfo.UseShellExecute = false;
                LocalExecStartInfo.UserName = Environment.UserName;
                LocalExecStartInfo.Password = SecPass;
                LocalExecStartInfo.Domain = Environment.UserDomainName;
                LocalExecStartInfo.LoadUserProfile = true;
                LocalExecStartInfo.WorkingDirectory = Environment.CurrentDirectory;
                LocalExecStartInfo.CreateNoWindow = true;
                LocalExecStartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
                LocalExecStartInfo.RedirectStandardOutput = true;
                LocalExecStartInfo.RedirectStandardError = true;
                System.Diagnostics.Process LocalExecProcess = new System.Diagnostics.Process();
                LocalExecProcess = System.Diagnostics.Process.Start(LocalExecStartInfo);
                System.IO.StreamReader ReaderOutput = LocalExecProcess.StandardOutput;
                System.IO.StreamReader ReaderError = LocalExecProcess.StandardError;
                String StandardOutput = "";
                try
                {
                    while (!LocalExecProcess.HasExited)
                    {
                        StandardOutput += ReaderOutput.ReadToEnd();
                        StandardOutput += ReaderError.ReadToEnd();
                        LocalExecProcess.WaitForExit(TimeSpan.FromSeconds(1).Milliseconds);
                    }
                }
                catch(System.ComponentModel.Win32Exception) {}
                System.IO.File.WriteAllText(
                    System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecOutput.txt"),
                    StandardOutput);
```

```
                if (LocalExecProcess.ExitCode != 0)
                    System.IO.File.WriteAllText(
                        System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecExitCode.txt"),
                        LocalExecProcess.ExitCode.ToString());
            }
            catch (Exception e)
            {
                //Error messages are low priority
                try
                {
                    System.Diagnostics.StackTrace Stack = new System.Diagnostics.StackTrace(e);
                    System.Diagnostics.StackFrame Frame = Stack.GetFrame(Stack.FrameCount - 1);
                    Int32 Line = Frame.GetFileLineNumber();
                    System.IO.File.WriteAllText(
                        System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecExitCode.txt"),
                        e.Message + " " + e.GetType().FullName + " " + Line);
                }
                catch (System.IO.IOException)
                { }
            }
            //Most of these settings have little bearing on access denied error messages
        }
        /// <summary>
        /// This compiles a service for remote execution
        /// </summary>
        ///
        public void CompileService()
        {
            if
            (Environment.MachineName.Equals(Environment.UserDomainName,StringComparison.CurrentCultureIgnoreCase)
            )
            {
                //CompileService("Environment.MachineName", Environment.UserName, Tscan.Scan.Password);
                CompileService("Environment.MachineName", "\"" + Environment.UserName + "\"",
                Tscan.Scan.Password);
            }
            else
            {
                CompileService("\"" + Environment.UserDomainName + "\"", "\"" + Environment.UserName + "\"",
                Tscan.Scan.Password);
            }
        }
        /// <summary>
        /// This compiles a service for remote execution
        /// </summary>
        ///
        public void CompileService(String Domain, String User, String Pass)
        {
            ServiceTextToCompile = ServiceTextToCompile.Replace("RemoteExecScriptString", RemoteExecScript);
            ServiceTextToCompile = ServiceTextToCompile.Replace("PasswordString", Pass);
            ServiceTextToCompile = ServiceTextToCompile.Replace("\"UserNameString\"", User);
            ServiceTextToCompile = ServiceTextToCompile.Replace("\"DomainNameString\"", Domain);
            try
            {
                System.IO.File.WriteAllText(
                    System.IO.Path.Combine(Environment.CurrentDirectory, "Service.cs"), ServiceTextToCompile);
            }
            catch { }

            Microsoft.CSharp.CSharpCodeProvider provider =
                new Microsoft.CSharp.CSharpCodeProvider();
            System.CodeDom.Compiler.CompilerParameters parameters =
                new System.CodeDom.Compiler.CompilerParameters();
            //parameters.CompilerOptions = "/unsafe";
            parameters.ReferencedAssemblies.Add("System.dll");
            //parameters.ReferencedAssemblies.Add("System.Diagnostics.dll");//Doesn't exist
            parameters.ReferencedAssemblies.Add("System.ServiceProcess.dll");
            parameters.ReferencedAssemblies.Add("System.Windows.Forms.dll");
            parameters.GenerateInMemory = false;
            parameters.GenerateExecutable = true;
            parameters.OutputAssembly = "TempService.exe";
            System.CodeDom.Compiler.CompilerResults results =
                provider.CompileAssemblyFromSource(parameters, ServiceTextToCompile);
            if (results.Errors.HasErrors)
            {
                System.Windows.Forms.MessageBox.Show(results.Errors[0].ErrorText);
                return;
            }
            PathToService = results.PathToAssembly;
            //System.Reflection.Assembly Assembly = results.CompiledAssembly;
            //Type program = Assembly.GetType("First.Program");
            //System.Reflection.MethodInfo main = program.GetMethod("Main");
            //main.Invoke(null, null);
```

```
1553            }
1554            /// <summary>
1555            /// This creates a service using remote service. This is legacy code.
1556            /// </summary>
1557            ///
1558            public void CreateServiceUsingRemoteService(String Server, String LocalDirectory, String User, String
                Pass, String Domain, String Suffix)
1559            {
1560                if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase)) Domain = ".";
1561                IntPtr Handle;
1562                if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
1563                {
1564                    Handle = OpenSCManager(null, null, 0xF003Fu);
1565                    //sc_manager_all_access = 0xF003Fu truecrypt uses this hex
1566                }
1567                else
1568                {
1569                    Handle = OpenSCManager(Server, null, 0xF003Fu);
1570                }
1571                if (Handle.Equals(IntPtr.Zero))
1572                {
1573                    System.Windows.Forms.MessageBox.Show(Marshal.GetLastWin32Error().ToString());
1574                    //I get a lot of 5 access denied fixed by app.manifest requireAdministrator
1575                }
1576                else
1577                {
1578                    IntPtr serviceHandle = CreateService(Handle, "Temp"+Suffix, "Temp Service", 0xF01FFu, 0x10u,
1579                        0x3u, 0x1u, LocalDirectory + "\\TempService.exe", null, null, null, User + "@" + Domain,
                            Pass); //Domain + "\\" + User, Pass);
1580                    //Service_All_Access = 0xF01FFu truecrypt uses this hex
1581                    if (serviceHandle.Equals(IntPtr.Zero))
1582                    {
1583                        System.Windows.Forms.MessageBox.Show("CreateService " +
                            Marshal.GetLastWin32Error().ToString());
1584                        //I get bad pointer due to failed openscmanager Error_Invalid_Handle 6
1585                        //I get Error_Invalid_Parameter 87 on lpdwTagId with "0" using null works
1586                        //1057 0x421 ERROR_INVALID_SERVICE_ACCOUNT
1587
                            //https://stackoverflow.com/questions/8811590/calling-createservice-when-explicitly-specifyin
                            g-the-local-domain-in-lpservicest
1588                        //1072 on failed service delete try restarting remote machine
1589                    }
1590                    else
1591                    {
1592                        CloseServiceHandle(serviceHandle);
1593                    }
1594                    CloseServiceHandle(Handle);
1595                }
1596            }
1597            /// <summary>
1598            /// This allows you to run a program on a remote system usually as a subset of a server pool
1599            /// </summary>
1600            ///
1601            public bool RemoteExec(String Server, String Domain, String User, String Pass)
1602            {
1603                //powershell -executionpolicy bypass -command \"& {$env:username >> test.txt}\"
1604                //C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -executionpolicy bypass -command
                    \"& {}\"
1605                //powershell remote script issues chaining to uac available as runas
1606                //cpuz_x32.exe -txt=cpuz.txt
1607                //requires uac elevation which is available as runas
1608                //local service limitations on write
1609                if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase)) return false;
1610                String RemoteDirectory = "c:\\windows";
1611                String LocalDirectory = RemoteDirectory + "\\TempService";
1612                if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
1613                {
1614                    //RemoteDirectory = RemoteDirectory + "\\TempService";
1615                    //admin$ doesn't exist for local machine without a network
1616                    LocalExec(Server, User, Pass);
1617                    return true;
1618                }
1619                else if (RemoteDirectory.Contains("c:\\windows"))
1620                {
1621                    RemoteDirectory = "\\\\" + Server + "\\" + RemoteDirectory.Replace("c:\\windows", "admin$") +
1622                        "\\TempService";
1623                }
1624                else
1625                {
1626                    RemoteDirectory = "\\\\" + Server + "\\" + RemoteDirectory.Replace(":", "$") + "\\TempService";
1627                }
1628                System.Management.ManagementScope Scope = Tscan.Scan.SetupScope(Server, Domain, User, Pass);
1629                try
```

```
{
    Scope.Connect();
}
catch
{
    return false;
}
String Suffix = ""; new System.Random().Next(1, 999).ToString();
//New instances
System.Management.ManagementPath Path =
    new System.Management.ManagementPath("Win32_Service.Name='" + Suffix + "'");
//System.Management.ManagementObject Obj;
System.Management.ManagementBaseObject OutParams;
uint ReturnValue = 0;

//Stop service old
SingleWMIMethod(Scope, "StopService", Suffix);
System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));

//Delete service old
SingleWMIMethod(Scope, "Delete", Suffix);
System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));

System.Net.NetworkCredential Cred = new System.Net.NetworkCredential(Domain + "\\" + User, Pass);
System.Net.CredentialCache Cache = new System.Net.CredentialCache();
Cache.Add(new System.Uri("\\\\" + Server + "\\admin$\\"), "basic", Cred);
//basic, digest, ntlm, kerberos

//error on nonexistance
if (System.IO.Directory.Exists(RemoteDirectory))
    System.IO.Directory.Delete(RemoteDirectory, true);
System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));

try
{
    System.IO.Directory.CreateDirectory(RemoteDirectory);
}
catch (System.IO.IOException)
{
    //MessageBox.Show(e.Message);
    return false;
}
if (!RemoteExecScript.ToLower().Contains("Powershell".ToLower()))
{
    System.IO.File.Copy(Environment.CurrentDirectory + "\\" + RemoteExecScript.Split("
    ".ToCharArray())[0],
        RemoteDirectory + "\\" + RemoteExecScript.Split(" ".ToCharArray())[0], true);
}
Boolean Caught = true;
for (Int16 i = 0; i < 3 && Caught; i++)
{
    if (i > 0) System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
    Caught = false;
    try
    {
        //copy fails 1/2 the time in .net 3.5 with IOException
        System.IO.File.Copy(PathToService, RemoteDirectory + "\\TempService.exe", true);
    }
    catch (System.IO.IOException)
    {
        //MessageBox.Show(e.Message);
        Caught = true;
    }
}
//try
//{
//    System.IO.File.Copy(PathToService, RemoteDirectory + "\\TempService.exe", true);
//}
//catch { }
//create service new
System.Management.ManagementClass ServiceManagementClass =
    new System.Management.ManagementClass(Scope, new
    System.Management.ManagementPath("Win32_Service"),new System.Management.ObjectGetOptions());
System.Management.ManagementBaseObject InParams =
ServiceManagementClass.GetMethodParameters("Create");
InParams["Name"] = "Temp" + Suffix;
InParams["DisplayName"] = "Temp Service";
InParams["ServiceType"] = 16;  //16, own process
InParams["ErrorControl"] = 0;  //0, hide errors, 1, show errors
InParams["StartMode"] = "Automatic";
InParams["DesktopInteract"] = false; //previous incorrect false
InParams["PathName"] = LocalDirectory + "\\TempService.exe";
```

```
1710            //CreateServiceUsingRemoteService(Server, LocalDirectory,User, Pass, Domain, "");
1711
1712            //http://www.pinvoke.net/default.aspx/advapi32.CreateService
1713            //System.ServiceProcess.ServiceController[] Services =
1714            //  System.ServiceProcess.ServiceController.GetServices(Server);
1715
1716            //hService = ::CreateService(
1717            //            hSCM, remoteServiceName, remoteServiceName,
1718            //            SERVICE_ALL_ACCESS, //0xF01FF, access right winsvc.h, DesiredAccess, c++ option
1719            //            serviceType,
1720            //            SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
1721            //            svcExePath,
1722            //            NULL, NULL,
1723            //            NULL, NULL ); //using LocalSystem
1724            //
                https://stackoverflow.com/questions/25619112/how-do-i-fix-the-error1069-the-service-did-not-start-due
                -to-logon-failure
1725            //InParams["StartName"] = User + "@" + Domain;
1726            InParams["StartName"] = Domain.ToLower() + "\\" + User.ToLower();
1727            InParams["StartPassword"] = Pass;
1728            //InParams["LoadOrderGroup"]
1729            //InParams["LoadOrderGroupDependencies[]"]
1730            //InParams["ServiceDependencies[]"]
1731            System.Management.InvokeMethodOptions methodOptions =
1732                new System.Management.InvokeMethodOptions(null, System.TimeSpan.FromMinutes(5));
1733            OutParams = ServiceManagementClass.InvokeMethod("Create", InParams, methodOptions);
1734            ReturnValue = System.Convert.ToUInt32(OutParams.Properties["ReturnValue"].Value);
1735            if (ReturnValue != 0) MessageBox.Show("Win32_Service.Create " + ReturnValue);
1736            // 8 means interactive process InParams["DesktopInteract"] = true; Administrator@Computer but not
                really
1737
1738            //CreateServiceUsingRemoteService(Server, LocalDirectory, User, Pass, Domain, Suffix);
1739
1740            if(true)
1741            {
1742                Path = new System.Management.ManagementPath("Win32_Service.Name='Temp" + Suffix + "'");
1743                System.Management.ManagementObject Obj = new System.Management.ManagementObject(Scope, Path,
                    new System.Management.ObjectGetOptions());
1744                InParams = Obj.GetMethodParameters("Change");
1745                //InParams["StartName"] = User + "@" + Domain;
1746                InParams["StartName"] = Domain.ToLower() + "\\" + User.ToLower();
1747                InParams["StartPassword"] = Pass;
1748                methodOptions = new System.Management.InvokeMethodOptions(null, System.TimeSpan.FromMinutes(5));
1749                OutParams = Obj.InvokeMethod("Change", InParams, methodOptions);
1750                ReturnValue = System.Convert.ToUInt32(OutParams.Properties["ReturnValue"].Value);
1751                if (ReturnValue != 0) MessageBox.Show("Win32_Service.Change " + ReturnValue);
1752                //code 8 on user@domain
1753            }
1754
1755            //Start service
1756            SingleWMIMethod(Scope, "StartService", Suffix);
1757
1758            //Return code 13 The service failed to find the service needed from a dependent service
1759            //startname = Administrator
1760            //Return code 15 failed authentication when using startname = local admins
1761            //Access is denied
1762            //at System.Diagnostics.Process.StartWithCreateProcess(ProcessStartInfo startInfo)
1763            //at System.Diagnostics.Process.Start(ProcessStartInfo startInfo)
1764            //at WindowsService.WindowsService.OnStart(String[] args)
1765
1766            System.Threading.Thread.Sleep(TimeSpan.FromMinutes(10));
1767            //this should be enough for most patches and installers on /qb
1768            //add 1 minutes for stop and delete twice
1769            //the program may run after the service has stopped
1770            SingleWMIMethod(Scope, "StopService", Suffix);
1771            System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));
1772
1773            SingleWMIMethod(Scope, "Delete", Suffix);
1774            System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));
1775
1776            System.IO.Directory.CreateDirectory(Environment.CurrentDirectory + "\\" + Server);
1777
1778            foreach (String SourceFile in System.IO.Directory.GetFiles(RemoteDirectory))
1779            {
1780                System.IO.File.Copy(SourceFile,
1781                    Environment.CurrentDirectory + "\\" + Server + "\\" +
1782                    System.IO.Path.GetFileName(SourceFile), true);
1783            }
1784            if (System.IO.File.Exists(System.IO.Path.Combine(
1785                System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecExitCode.txt")))
1786                System.IO.File.Copy(System.IO.Path.Combine(
1787                System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecExitCode.txt"),
1788                Environment.CurrentDirectory + "\\" + Server + "\\" + "RemoteExecExitCode.txt", true);
```

```
            if(System.IO.File.Exists(System.IO.Path.Combine(
                System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecOutput.txt")))
                System.IO.File.Copy(System.IO.Path.Combine(
                System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecOutput.txt"),
                Environment.CurrentDirectory + "\\" + Server + "\\" + "RemoteExecOutput.txt", true);
            System.IO.Directory.Delete(RemoteDirectory, true);

            //MessageBox.Show("RemoteExec Server Done: " + Server);
            return true;
        }
        /// <summary>
        /// Run one wmi method
        /// </summary>
        ///
        public UInt32 SingleWMIMethod(System.Management.ManagementScope Scope, String Method, String Suffix)
        {
            UInt32 ReturnValue = 0;
            try
            {
                System.Management.ManagementPath Path =
                    new System.Management.ManagementPath("Win32_Service.Name='Temp"+Suffix+"'");
                System.Management.ManagementObject Obj =
                    new System.Management.ManagementObject(Scope, Path, new
                    System.Management.ObjectGetOptions());
                System.Management.ManagementBaseObject OutParams =
                    Obj.InvokeMethod(Method, (System.Management.ManagementBaseObject)null, null);
                ReturnValue = System.Convert.ToUInt32(OutParams.Properties["ReturnValue"].Value);
                if (ReturnValue != 0) MessageBox.Show("Win32_Service." + Method + " " + ReturnValue);
            }
            catch { }
            return ReturnValue;
        }
    }
    public class ScannerActiveDirectory
    {
        public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainAdminList;
        public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainList;
        public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainInProgressList;
        //public System.Collections.Specialized.StringDictionary DomainAdminList;
        //public System.Collections.Specialized.StringDictionary DomainList;
        //public System.Collections.Specialized.StringDictionary DomainInProgressList;
        /// <summary>
        /// This instantiates a Active Directory scanning object
        /// </summary>
        ///
        public ScannerActiveDirectory()
        {
            DomainAdminList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
            DomainList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
            DomainInProgressList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
        }
        /// <summary>
        /// This builds a server list from Active Directory
        /// </summary>
        ///
        public void ScanActiveDirectory()
        {
            int[] MaxThreads = { 0, 0 };
            int[] AvailableThreads = { 0, 0 };
            DomainList.TryAdd(Environment.UserDomainName, "0");
            Int32 OldDomainCount = 0;
            for (Int16 i = 0; i < 10; i++)
            {
                Boolean Quit = true;
                foreach (String Domain in DomainList.Keys)
                {
                    if (!DomainList[Domain].Equals("Done", StringComparison.CurrentCultureIgnoreCase))
                        Quit = false;
                }
                if (Quit && i >= 2) break;
                for (Int16 j = 0; j < 10; j++)
                {
                    OldDomainCount = DomainList.Count;
                    foreach (String Domain in DomainList.Keys)
                    {
                        System.Threading.ThreadPool.QueueUserWorkItem(
                            new System.Threading.WaitCallback(ScanTrusts), Domain);
                    }//foreach domain get trusts
                    System.Threading.ThreadPool.GetMaxThreads(
                        out MaxThreads[0], out MaxThreads[1]);
                    System.Threading.ThreadPool.GetAvailableThreads(
                        out AvailableThreads[0], out AvailableThreads[1]);
                    for (Int16 k = 0;
```

```
                        k < 10 && MaxThreads[0] - AvailableThreads[0] > 2 && DomainInProgressList.Count > 0;
                        k++)
                    {
                        Tscan.Scan.UpdateProgress("10 Minute. " + k + "/10");
                        System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
                        System.Threading.ThreadPool.GetMaxThreads(
                            out MaxThreads[0], out MaxThreads[1]);
                        System.Threading.ThreadPool.GetAvailableThreads(
                            out AvailableThreads[0], out AvailableThreads[1]);
                    }
                    if (OldDomainCount == DomainList.Count) break;
                }//for up to 10 trust scans
                foreach (String Domain in DomainList.Keys)
                {
                    if (!DomainInProgressList[Domain].Equals(
                        "InProgress", StringComparison.CurrentCultureIgnoreCase) &&
                        !DomainList[Domain].Equals(
                        "Done", StringComparison.CurrentCultureIgnoreCase) &&
                        !DomainList[Domain].Equals(
                        "10", StringComparison.CurrentCultureIgnoreCase))
                    {
                        Tscan.Scan.UpdateProgress(Domain);
                        System.Threading.ThreadPool.QueueUserWorkItem(
                            new System.Threading.WaitCallback(ScanDomain), Domain);
                    }
                }
                System.Threading.ThreadPool.GetMaxThreads(
                    out MaxThreads[0], out MaxThreads[1]);
                System.Threading.ThreadPool.GetAvailableThreads(
                    out AvailableThreads[0], out AvailableThreads[1]);
                for (Int16 j = 0;
                    j < 60  && MaxThreads[0] - AvailableThreads[0] > 2 && DomainInProgressList.Count > 0;
                    j++)
                {
                    Tscan.Scan.UpdateProgress("1 Hour Wait. " + j + "/60");
                    System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
                    System.Threading.ThreadPool.GetMaxThreads(
                        out MaxThreads[0], out MaxThreads[1]);
                    System.Threading.ThreadPool.GetAvailableThreads(
                        out AvailableThreads[0], out AvailableThreads[1]);
                }
            }//for up to 10 loops of both trusts and undone domains
            //AD only stub
            System.IO.File.WriteAllText("Domainlist.txt", "");
            foreach (String Key in DomainList.Keys)
                System.IO.File.AppendAllText("Domainlist.txt", Key + Environment.NewLine);
        }
        /// <summaryScanWinsAndSql
        /// This builds a list of servers from wins
        /// </summary>
        ///
        public void ScanWinsAndSql()
        {
            String StringTable = "";
            String StringHeader = "";
            String StringRow = "";
            Boolean HeaderDone = false;
            System.DirectoryServices.DirectoryEntry Root = new System.DirectoryServices.DirectoryEntry("WinNT:");
            foreach (System.DirectoryServices.DirectoryEntry Workgroup in Root.Children)
            {
                String WorkgroupName = Workgroup.Name;
                foreach (System.DirectoryServices.DirectoryEntry Computer in Workgroup.Children)
                {
                    String ComputerName = Computer.Name;
                    if (!ComputerName.Equals("Schema", StringComparison.CurrentCultureIgnoreCase))
                    {
                        StringRow = "\"" + ComputerName + "\",\"" + WorkgroupName + "\",";
                        StringHeader = "Computer,\"Workgroup\",";
                        try
                        {
                            foreach (String Name in Computer.Properties.PropertyNames)
                            {
                                try
                                {
                                    StringRow += "\"" + Computer.Properties[Name].Value + "\",";
                                }
                                catch
                                {
                                    StringRow += "\"\",";
                                }
                                StringHeader += "\"" + Name + "\",";
                            }
                        }
```

```
                        catch { } //sometimes there are no properties I think
                        if (!HeaderDone) StringTable = StringHeader + Environment.NewLine;
                        HeaderDone = true;
                        StringTable += StringRow + Environment.NewLine;
                        Tscan.Scan.ServerList.TryAdd(ComputerName, "");
                    }
                }
            }
            Tscan.Scan.WriteToDisk("Wins.csv", StringTable);
            StringTable = "Computer,\"Instance\",\"Clustered\",\"Version\"," + Environment.NewLine;
            System.Data.DataTable Table = System.Data.Sql.SqlDataSourceEnumerator.Instance.GetDataSources();
            foreach (System.Data.DataRow Row in Table.Rows)
            {
                StringTable += "\"" + Row.ItemArray[0].ToString() + "\"," +
                    "\"" + Row.ItemArray[1].ToString() + "\"," +
                    "\"" + Row.ItemArray[2].ToString() + "\"," +
                    "\"" + Row.ItemArray[3].ToString() + "\"," + Environment.NewLine;
                //TryAdd
                if (!String.IsNullOrEmpty(Row.ItemArray[0].ToString()) &&
                    !Tscan.Scan.ServerList.ContainsKey(Row.ItemArray[0].ToString()))
                    Tscan.Scan.ServerList.TryAdd(Row.ItemArray[0].ToString(), "");
            }
            Tscan.Scan.WriteToDisk("SQL.csv", StringTable);
        }
        /// <summary>
        /// This builds a list of admins with bad passwords
        /// </summary>
        ///
        public void ScanTrusts(Object ObjectDomain)
        {
            String Domain = ObjectDomain.ToString();
            DomainInProgressList.TryAdd(Domain, "InProgress");
            try
            {
                Tscan.Scan.UpdateProgress(Domain);
                System.DirectoryServices.ActiveDirectory.Domain DomainDomain =
                    System.DirectoryServices.ActiveDirectory.Domain.GetDomain(
                    new System.DirectoryServices.ActiveDirectory.DirectoryContext(
                        System.DirectoryServices.ActiveDirectory.DirectoryContextType.Domain, Domain));
                foreach (System.DirectoryServices.ActiveDirectory.Domain Child in DomainDomain.Children)
                {
                    if (!DomainList.ContainsKey(Child.Name))DomainList.TryAdd(Child.Name, "0");
                }
                foreach (System.DirectoryServices.ActiveDirectory.TrustRelationshipInformation Trust in
                    DomainDomain.GetAllTrustRelationships())
                {
                    if (!DomainList.ContainsKey(Trust.SourceName))DomainList.TryAdd(Trust.SourceName, "0");
                    if (!DomainList.ContainsKey(Trust.TargetName))DomainList.TryAdd(Trust.TargetName, "0");
                }
                if (!DomainList.ContainsKey(DomainDomain.Parent.Name))
                    DomainList.TryAdd(DomainDomain.Parent.Name, "0");
                //System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
            }
            catch { }
            String Skip = "";
            DomainInProgressList.TryRemove(Domain, out Skip);
        }
        /// <summary>
        /// This builds a list of admins with bad passwords
        /// </summary>
        ///
        public void ScanAdmins()
        {
            Boolean Self = false;
            System.DirectoryServices.ActiveDirectory.Domain UserDomain =
                System.DirectoryServices.ActiveDirectory.Domain.GetCurrentDomain();
            while (UserDomain.Parent != null) UserDomain = UserDomain.Parent;
            String DomainDN = UserDomain.GetDirectoryEntry().Properties["distinguishedName"].ToString()
                .Replace("OU=Domain Controllers,","");
            String StringFilter = "(&(objectClass=User))(memberOf=CN=Domain Admins,CN=Users,"+DomainDN+"))";
            String[] StringProperties = {"Name"};
            System.DirectoryServices.ActiveDirectory.Domain DomainDomain = UserDomain;
            System.DirectoryServices.DirectorySearcher Finder =
                new System.DirectoryServices.DirectorySearcher(
                    DomainDomain.GetDirectoryEntry(), StringFilter, StringProperties);
            Finder.ClientTimeout = TimeSpan.FromMinutes(1);
            Finder.Asynchronous = true;
            Finder.PageSize = 1000;
            Finder.ServerPageTimeLimit = TimeSpan.FromMinutes(1);
            Finder.ServerTimeLimit = TimeSpan.FromMinutes(1);
            foreach (System.DirectoryServices.SearchResult Row in Finder.FindAll())
            {
                String Name = "Name";
```

```
                if (String.IsNullOrEmpty(Row.Properties[Name].ToString()))
                {
                    foreach (String Pass in Tscan.Scan.WMIPasswords)
                    {
                        if (Tscan.Scan.TestPassword(
                            UserDomain.FindDomainController().Name,
                            UserDomain.Name,
                            Row.Properties[Name].ToString(),
                            Pass).Equals(1))
                        {
                            DomainAdminList.TryAdd(
                                UserDomain.Name + "\\" + Row.Properties[Name].ToString(),
                                Pass);
                            if (UserDomain.Name.Equals(
                                Environment.UserDomainName, StringComparison.CurrentCultureIgnoreCase) &&
                                Row.Properties[Name].ToString().Equals(
                                Environment.UserName, StringComparison.CurrentCultureIgnoreCase))
                                Self = true;
                        }
                    }
                }
            }
            if (!Self)
            {
                String Current = DomainAdminList.Keys.GetEnumerator().Current.ToString();
                String Dom = Current.Split("\\".ToCharArray())[0];
                String User = Current.Split("\\".ToCharArray())[1];
                String Pass = DomainAdminList[Current];
                Tscan.Scan.RemoteExec.CompileService("\"" + Dom + "\"", User, Pass);
            }

        }
        /// <summary>
        /// This scans a single domain
        /// </summary>
        ///
        public void ScanDomain(Object ObjectDomain)
        {
            String Domain = ObjectDomain.ToString();
            DomainInProgressList.TryAdd(Domain, "InProgress");
            ScanDomain(Domain,"Computer");
            ScanDomain(Domain,"User");
            String Skip = "";
            DomainInProgressList.TryRemove(Domain, out Skip);
        }
        /// <summary>
        /// This scans a single domain and a single type of object
        /// </summary>
        ///
        public void ScanDomain(String Domain, String Object)
        {
            //String StringDomain = Domain.ToString();
            String Names = "";
            String Values = "";
            String Table = "";
            Boolean HeaderDone = false;
            Boolean Fail = false;
            String StringFilter = "";
            String[] StringProperties;
            String[] StringPropertiesComputer = {"name","StreetAddress","PhysicalDeliveryOfficeName","l","st",
                    "PostalCode","co","TelephoneNumber","mail",
                    "DNSHostName","MacAddress","OperatingSystem","Description"};
            String[] StringPropertiesUser ={ "name","StreetAddress","PhysicalDeliveryOfficeName","l","st",
                    "PostalCode","co","TelephoneNumber","mail",
                    "GivenName","sn","EmployeeID","LastLogon","LastLogonTimestamp"};
            if (Object.Equals("Computer", StringComparison.CurrentCultureIgnoreCase))
            {
                StringFilter = "(objectClass=Computer)";
                StringProperties = StringPropertiesComputer;
                //File size for 200k users is 20MB
            }
            else if (Object.Equals("User", StringComparison.CurrentCultureIgnoreCase))
            {
                StringFilter = "(objectClass=User)";
                StringProperties = StringPropertiesUser;
                //mostrecentuser may be goose chase
            }
            else
            {
                IncrementDomainTries(Domain);
                return;
            }
            try
```

```
            {
                //System.ArguementException on getdomain
                System.DirectoryServices.ActiveDirectory.Domain DomainDomain =
                    System.DirectoryServices.ActiveDirectory.Domain.GetDomain(
                    new System.DirectoryServices.ActiveDirectory.DirectoryContext(
                        System.DirectoryServices.ActiveDirectory.DirectoryContextType.Domain, Domain));
                System.DirectoryServices.DirectorySearcher Finder =
                    new System.DirectoryServices.DirectorySearcher(
                        DomainDomain.GetDirectoryEntry(), StringFilter, StringProperties);
                Finder.ClientTimeout = TimeSpan.FromMinutes(1);
                Finder.Asynchronous = true;
                Finder.PageSize = 1000;
                Finder.ServerPageTimeLimit = TimeSpan.FromMinutes(1);
                Finder.ServerTimeLimit = TimeSpan.FromMinutes(1);
                foreach (System.DirectoryServices.SearchResult Row in Finder.FindAll())
                {
                    foreach (String Name in Row.Properties.PropertyNames)
                    {
                        Names += "\"" + Name + "\",";
                        if (Row.Properties[Name] != null)
                        {
                            Values += "\"" + Tscan.Scan.ScrubString(Row.Properties[Name].ToString()) + "\",";
                        }
                        else
                        {
                            Values += "\"\",";
                        }

                        //String foo = Row.Properties[Name].ToString();
                    }
                    if (Object.Equals("Computer", StringComparison.CurrentCultureIgnoreCase))
                    {
                        if (String.IsNullOrEmpty(Row.Properties["DNSHostName"].ToString()))
                            Tscan.Scan.ServerList.TryAdd(Row.Properties["Name"].ToString(), "");
                        else Tscan.Scan.ServerList.TryAdd(Row.Properties["DNSHostName"].ToString(), "");
                    }
                }
            }
            catch
            {
                IncrementDomainTries(Domain);
                Fail = true;
            }
            if (!HeaderDone) Table += Names + Environment.NewLine;
            Table += Values + Environment.NewLine;
            HeaderDone = true;
            Tscan.Scan.WriteToDisk(Domain + "_" + Object + ".csv", Table);
            if (String.IsNullOrEmpty(Table)) IncrementDomainTries(Domain);
            if (!String.IsNullOrEmpty(Table) && !Fail) DomainList[Domain] = "Done";
        }
        /// <summary>
        /// This helps track the number of times a domain has thrown an error
        /// </summary>
        ///
        public void IncrementDomainTries(String Domain)
        {
            Int16 Tries = 0;
            if (Int16.TryParse(DomainList[Domain], out Tries))
            {
                if (Tries < 10)
                {
                    Tries++;
                    DomainList[Domain] = Tries.ToString();
                }
            }
        }
    }
}
```