

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Windows.Forms;
5  using System.Runtime.InteropServices;
6
7  namespace Tscan
8  {
9      static class Tscan
10     {
11         /// <summary>
12         /// The main entry point for the application.
13         /// </summary>
14         ///
15         public static Scanner Scan;
16         [MTAThread]
17         static void Main(String[] args)
18         {
19             System.Threading.ThreadPool.SetMinThreads(10, 10);
20             System.Threading.ThreadPool.SetMaxThreads(Environment.ProcessorCount * 20, 10);
21             if (args.Count() == 0)
22             {
23                 Application.EnableVisualStyles();
24                 Application.SetCompatibleTextRenderingDefault(false);
25                 Application.Run(new ScanType());
26             }
27             else
28             {
29                 Scan = new Scanner();
30                 if (args.Count() > 0)
31                 {
32                     foreach (String arg in args)
33                     {
34                         if (arg.StartsWith("//Type", StringComparison.CurrentCultureIgnoreCase))
35                         {
36                             if (!int.TryParse(arg.Split(":").ToCharArray()[1], out Scan.IntScanType))
37                             {
38                                 if (arg.Split(":").ToCharArray()[1].Equals("ThisMachine",
39                                     StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 1;
40                                 else if (arg.Split(":").ToCharArray()[1].Equals("ServerList",
41                                     StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 2;
42                                 else if (arg.Split(":").ToCharArray()[1].Equals("Subnet",
43                                     StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 3;
44                                 else if (arg.Split(":").ToCharArray()[1].Equals("ActiveDirectory",
45                                     StringComparison.CurrentCultureIgnoreCase)) Scan.IntScanType = 4;
46                             }
47                         }
48                         else if (arg.StartsWith("//File", StringComparison.CurrentCultureIgnoreCase))
49                         {
50                             Scan.ServerListFilename = arg.Split(":").ToCharArray()[1];
51                         }
52                         else if (arg.StartsWith("//Password", StringComparison.CurrentCultureIgnoreCase))
53                         {
54                             Scan.Password = arg.Split(":").ToCharArray()[1];
55                         }
56                         else if (arg.StartsWith("//SearchTerms", StringComparison.CurrentCultureIgnoreCase))
57                         {
58                             Scan.SearchTerm = arg.Split(":").ToCharArray()[1];
59                         }
60                         else if (arg.StartsWith("//Script", StringComparison.CurrentCultureIgnoreCase))
61                         {
62                             Scan.RemoteExec.RemoteExecScript = arg.Split(":").ToCharArray()[1];
63                         }
64                         else if (arg.StartsWith("//WMIObjects", StringComparison.CurrentCultureIgnoreCase))
65                         {
66                             Scan.SearchObjects = arg.Split(":").ToCharArray()[1];
67                         }
68                         else if (arg.StartsWith("//ADOnly", StringComparison.CurrentCultureIgnoreCase))
69                         {
70                             Scan.ADOnly = true; ;
71                         }
72                         else if (arg.StartsWith("//Internet", StringComparison.CurrentCultureIgnoreCase))
73                         {
74                             Scan.ScanInternet = true; ;
75                         }
76                     }
77                     Scan.ScanNetQueueWorkItem("");
78                 }
79             }
80         }
81     }
82
83     public class Scanner

```

```

84 {
85     [DllImport("IPHLPAPI.DLL", ExactSpelling = true)]
86     public static extern int SendARP(uint DestIP, uint SrcIP, byte[] pMacAddr, ref uint PhyAddrLen);
87
88     public int IntScanType;
89     public Boolean ADOnly;
90     public Boolean ScanInternet;
91     Int32 IntDone;
92     Int32 IntSQL;
93     Int32 IntPort;
94     Int32 IntServer;
95     public String Password;
96     public String SearchTerm;
97     public String SearchObjects;
98     public String MACLookupURI;
99     public String[] WMIPasswords;
100    public String[] WMIUsernames;
101    public String[] XMLElements;
102    public String ServerListFilename;
103    //public System.Collections.Specialized.StringDictionary ServerList;
104    public System.Collections.Concurrent.ConcurrentDictionary<String, String> ServerList;
105    public ScannerActiveDirectory ScanAD;
106    public ScannerRemoteExec RemoteExec;
107    public Progress ProgressForm;
108    /// <summary>
109    /// Constructor for the main object
110    /// </summary>
111    ///
112    public Scanner()
113    {
114        IntScanType = 0;
115        ADOnly = false;
116        IntDone = 0;
117        IntSQL = 0;
118        IntPort = 0;
119        ServerListFilename = "";
120        ServerList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
121        ScanAD = new ScannerActiveDirectory();
122        RemoteExec = new ScannerRemoteExec();
123        String[] WMIPasswordsTemp = {Password, "password", "123456", ""};
124        WMIPasswords = WMIPasswordsTemp;
125        //SplashData 4+2% limited to avoid account lockout
126        String[] WMIUsernamesTemp = {Environment.UserName, "administrator",
127            "Administrator", "user1", "admin", "demo", "db2admin", "Admin", "sql"};
128        WMIUsernames = WMIUsernamesTemp;
129        //rapid7 several are 1/10k
130    }
131    /// <summary>
132    /// This writes a file to disk
133    /// </summary>
134    ///
135    public void WriteToDisk(String Title, String Table)
136    {
137        Boolean Caught = true;
138        for (Int16 i = 0; i < 3 && Caught; i++)
139        {
140            if (i > 0) System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
141            Caught = false;
142            try
143            {
144                System.IO.File.WriteAllText(System.IO.Path.Combine(
145                    Environment.CurrentDirectory, Title), Table);
146            }
147            catch (System.IO.IOException e)
148            {
149                MessageBox.Show(e.Message);
150                Caught = true;
151            }
152        }
153    }
154    /// <summary>
155    /// The setup for the threadpool and a new thread for the scanner
156    /// </summary>
157    ///
158    public void ScanNetQueueWorkItem(String ProgressLabel)
159    {
160        //MACLookup("");
161        System.Threading.ThreadPool.QueueUserWorkItem(
162            new System.Threading.WaitCallback(ScanNetWorkItem), ProgressLabel);
163    }
164    /// <summary>
165    /// This adjusts the size of the thread pool to fit the processor and memory. Deprecated.
166    /// </summary>

```

```

167     ///
168     public Boolean AreResourcesAvailable()
169     {
170         int[] MaxThreads = { 0, 0 };
171         System.Threading.ThreadPool.GetMaxThreads(out MaxThreads[0], out MaxThreads[1]);
172         System.Diagnostics.PerformanceCounter CPUCounter;
173         System.Diagnostics.PerformanceCounter RamCounter;
174         CPUCounter = new System.Diagnostics.PerformanceCounter("Processor", "% Processor Time", "_Total");
175         RamCounter = new System.Diagnostics.PerformanceCounter("Memory", "Available MBytes");
176         float CPUCountAvg = 0;
177         for (int i = 0; i < 10; i++)
178         {
179             CPUCountAvg += CPUCounter.NextValue();
180             System.Threading.Thread.Sleep(TimeSpan.FromSeconds(1));
181             //10 seconds per 144 threads on 1 hour threads is 52k threads per hour
182             //This reduces 1 second spikes associated with application launches
183         }
184         if (CPUCountAvg / 10 > 80 && MaxThreads[0] > Environment.ProcessorCount * 20) //20% free CPU
185         {
186             MaxThreads[0] -= Environment.ProcessorCount * 10;
187             System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
188             return false;
189         }
190         //else if (RamCounter.NextValue() < 200 && MaxThreads[0] > Environment.ProcessorCount * 20) //200MB
191         //free
192         //{
193         //    MaxThreads[0] -= Environment.ProcessorCount * 10;
194         //    System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
195         //    return false;
196         //} //10MB per thread
197         else if (MaxThreads[0] < Environment.ProcessorCount * 200) //Maxes out in 10 seconds * 20 steps is
198         //3 min
199         {
200             MaxThreads[0] += Environment.ProcessorCount * 10;
201             System.Threading.ThreadPool.SetMaxThreads(MaxThreads[0], MaxThreads[1]);
202             return true;
203         }
204         else
205         {
206             return false;
207         }
208     }
209     /// <summary>
210     /// This method updates progress
211     /// </summary>
212     ///
213     public void CountProgress()
214     {
215         IntPort = 0;
216         IntDone = 0;
217         IntSQL = 0;
218         foreach (String File in System.IO.Directory.GetFiles(Environment.CurrentDirectory))
219         {
220             if (File.EndsWith("Ping.csv", StringComparison.CurrentCultureIgnoreCase))
221             {
222                 IntPort++;
223             }
224             else if (File.EndsWith("Done.txt", StringComparison.CurrentCultureIgnoreCase))
225             {
226                 IntDone++;
227             }
228             else if (File.EndsWith("SqlServiceAdvancedProperty.csv",
229                 StringComparison.CurrentCultureIgnoreCase))
230             {
231                 IntSQL++;
232             }
233         }
234     }
235     /// <summary>
236     /// This method updates progress
237     /// </summary>
238     ///
239     public void UpdateProgress(String CurrentComputer)
240     {
241         //CountProgress();
242         int[] MaxThreads = { 0, 0 };
243         System.Threading.ThreadPool.GetMaxThreads(out MaxThreads[0], out MaxThreads[1]);
244         int[] AvailableThreads = { 0, 0 };
245         System.Threading.ThreadPool.GetAvailableThreads(out AvailableThreads[0], out AvailableThreads[1]);
246         String UpdateProgress = "Computers completed: " + IntDone + Environment.NewLine +
247             "SQL computers completed: " + IntSQL + Environment.NewLine +
248             "Servers completed: " + IntServer + Environment.NewLine +
249             "Port scans completed: " + IntPort + Environment.NewLine +

```

```

247         "Currently working on: " + CurrentComputer + Environment.NewLine +
248         "Total computers: " + ServerList.Count + Environment.NewLine +
249         (IntScanType == 4 ? "Total domains: " + ScanAD.DomainList.Count + Environment.NewLine : "") +
250         "Concurrent work items: " + (MaxThreads[0] - AvailableThreads[0]) + "/" + MaxThreads[0];
251     Tscan.Scan.ProgressForm.ProgressLabel.Invoke((MethodInvoker) (() =>
252         Tscan.Scan.ProgressForm.ProgressLabel.Text = UpdateProgress));
253 }
254 /// <summary>
255 /// This method requests threaded scans for globs of 10 servers each
256 /// </summary>
257 ///
258 public void ScanNetWorkItem(Object ProgressLabel)
259 {
260     String[] SubsetServerList = new String[10];
261     //test[1] = "str";
262     SubsetServerList[0] = "Empty";
263     RemoteExec.CompileService();
264     BuildServerList();
265     String[] ServerListArray;
266     int[] AvailableThreads = { 0, 0 };
267     for (int k = 0; k < 15; k++)
268     {
269         ServerListArray = new String[ServerList.Count];
270         ServerList.Keys.CopyTo(ServerListArray, 0);
271         for (Int32 i = 0; i <= ServerListArray.Count(); i += 10)
272         {
273             SubsetServerList = new String[10];
274             for (int j = 0; j < 10 && i + j < ServerListArray.Count(); j++)
275             { SubsetServerList[j] = ServerListArray[i + j]; }
276             System.Threading.ThreadPool.QueueUserWorkItem(
277                 new System.Threading.WaitCallback(Tscan.Scan.ScanTenServers), SubsetServerList);
278             if (i % (Environment.ProcessorCount * 20) == 0) AreResourcesAvailable();
279             UpdateProgress(SubsetServerList[0]);
280             System.Threading.ThreadPool.GetAvailableThreads(
281                 out AvailableThreads[0], out AvailableThreads[1]);
282             while (AvailableThreads[0] == 0)
283             {
284                 System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
285                 AreResourcesAvailable();
286                 UpdateProgress(SubsetServerList[0]);
287                 System.Threading.ThreadPool.GetAvailableThreads(
288                     out AvailableThreads[0], out AvailableThreads[1]);
289             }
290         }
291         for (Int32 i = 0; i < 60 * 8; i++)
292         {
293             System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
294             AreResourcesAvailable();
295             UpdateProgress("8 Hour Wait. " + i + "/480 " + k + "/15");
296             if (IntDone >= ServerList.Count * 0.998) Application.Exit();
297         }
298         IntPort = 0;
299     }
300     //return 0;
301 }
302 /// <summary>
303 /// This scans ten servers
304 /// </summary>
305 ///
306 public void ScanTenServers(Object SubsetServerList)
307 {
308     //String[] str =
309     //String[] str = ((System.Collections.IEnumerable)SubsetServerList).Cast<object>()
310     //    .Select(x => (x!=null?x.ToString():null)).ToArray();
311     foreach (String Server in (System.Collections.IEnumerable)SubsetServerList)
312     {
313         ScanServer(Server);
314     }
315 }
316 /// <summary>
317 /// This tries to scan using HTTP Headers
318 /// </summary>
319 ///
320 public Boolean HTTPHeaderScan(String Server)
321 {
322     String[] Ports = { "80", "443", "8080", "8008", "8443" };
323     foreach (String Port in Ports)
324     {
325         String StringTable = "";
326         String StringHeader = "";
327         String StringRow = "";
328         Boolean HeaderDone = false;
329         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 +
383             "/admin$/notepad.exe");
384     System.Net.FileWebResponse Resp;
385     try
386     {
387         Resp = (System.Net.FileWebResponse)Req.GetResponse();
388         StringRow = "\"" + Server + "\", ";
389         StringHeader = "Computer, ";
390         foreach (String Key in Resp.Headers.AllKeys)
391         {
392             try
393             {
394                 StringRow += "\"" + Resp.Headers[Key] + "\", ";
395             }
396             catch
397             {
398                 StringRow += "\"\\\", ";
399             }
400             StringHeader += "\"" + Key + "\", ";
401         }
402         if (!HeaderDone) StringTable = StringHeader + Environment.NewLine;
403         HeaderDone = true;
404         StringTable += StringRow + Environment.NewLine;
405         Resp.Close();
406     }
407     catch (System.Net.WebException)
408     {
409         return true;
410     }
411     WriteToDisk(Server + "_SMBHeader.csv", StringTable);
412     return true;

```

```

412     }
413     /// <summary>
414     /// This icmp pings, syn pings 20 ports, and arp pings to get the MAC vendor for ping.csv
415     /// </summary>
416     ///
417     public Boolean PortScan(String Server)
418     {
419         //ICMP - Broadcast, IP, Mac
420         //Syn - Port, State, Service, Version
421         //Nbtstat - Name, User, Mac
422         //SMB - OS
423         String Header = "";
424         String Values = "";
425         String Table = "";
426         Byte[] MacAddressBytes;
427         String Mac;
428         Int16[] ListOfPorts = {53, //DNS
429                                80, //HTTP
430                                135, //WMI
431                                137,138,139, //NetBIOS
432                                389, //LDAP
433                                443, //HTTPS
434                                445, //SMB
435                                1433,1434, //SQL
436                                5060,5061, //SIP
437                                5431, //UPNP
438                                8008,8080,8443, //HTTP
439                                9100,9220 //PCL
440                                };
441         Header = "Computer,";
442         Values = "\"\" + Server + "\",\"";
443         try
444         {
445             System.Net.NetworkInformation.PingReply PR = new
446             System.Net.NetworkInformation.Ping().Send(Server);
447             Values += "\"\" + PR.RoundtripTime + "\",\"";
448             Header += "\"\" + \"PingTime\" + "\",\"";
449             Values += "\"\" + PR.Status + "\",\"";
450             Header += "\"\" + \"PingStatus\" + "\",\"";
451         }
452         catch (System.Net.NetworkInformation.PingException e)
453         {
454             Values += "\"\" + 0 + "\",\"";
455             Header += "\"\" + \"PingTime\" + "\",\"";
456             Values += "\"\" + e.Message + "\",\"";
457             Header += "\"\" + \"PingStatus\" + "\",\"";
458         }
459         foreach (Int16 IntPort in ListOfPorts)
460         {
461             Values += "\"\" + SinglePort(Server, IntPort) + "\",\"";
462             Header += "\"\" + IntPort + "\",\"";
463         }
464         Header += "\"Mac\", \"Details\", \"";
465         System.Net.IPAddress ipAddress = System.Net.IPAddress.Parse("0.0.0.0");
466         Boolean Fail = false;
467         try
468         {
469             Int16 Skip = 0x0;
470             if (Server.Split(".").ToCharArray().Count() == 4 &&
471                 Int16.TryParse(Server.Split(".").ToCharArray()[0], out Skip) &&
472                 Int16.TryParse(Server.Split(".").ToCharArray()[1], out Skip) &&
473                 Int16.TryParse(Server.Split(".").ToCharArray()[2], out Skip) &&
474                 Int16.TryParse(Server.Split(".").ToCharArray()[3], out Skip))
475             {
476                 //if server is IP
477                 ipAddress = System.Net.IPAddress.Parse(Server);
478             }
479             else
480             {
481                 //if server is hostname
482                 ipAddress = System.Net.Dns.GetHostEntry(Server).AddressList[0];
483             }
484         }
485         catch
486         {
487             Fail = true;
488         }
489         if (!Fail)
490         {
491             uint UIntAddress = BitConverter.ToUInt32(ipAddress.GetAddressBytes(), 0);
492             uint MacAddressByteLength = 6;
493             MacAddressBytes = new Byte[MacAddressByteLength];
494             int SendARPSuccess = SendARP(UIntAddress, 0, MacAddressBytes, ref MacAddressByteLength);

```

```

494         if (SendARPSuccess == 0)
495         {
496             Values += "\"";
497             Mac = "";
498             for (int i = 0; i < MacAddressByteLength; i++)
499             {
500                 if (!String.IsNullOrEmpty(MacAddressBytes[i].ToString("x2")))
501                     Mac += MacAddressBytes[i].ToString("x2");
502                 else Mac += "--";
503                 if (i < MacAddressByteLength - 1) Mac += ":";
504             }
505             Values += Mac;
506             Values += "\", ";
507             Values += "\"";
508             Values += MACLookup(Mac);
509             Values += "\", ";
510         }
511         else
512         {
513             Values += "\"ARP Fail\", \"\", ";
514         }
515     }
516     else
517     {
518         Values += "\"DNS Fail\", \"\", ";
519     }
520     Table = Header + Environment.NewLine + Values;
521     WriteToDisk(Server + "_Ping.csv", Table);
522     return true;
523 }
524 /// <summary>
525 /// This looks up a MAC to it's vendor only. Runs 200 times an hour.
526 /// </summary>
527 ///
528 public String MACLookup(String MAC)
529 {
530     //https://api.macvendors.co/'MAC'/xml
531     //https://api.macvendors.com/'MAC'
532     System.Net.HttpWebRequest Req =
533         (System.Net.HttpWebRequest)System.Net.HttpWebRequest.Create(MACLookupURI.Replace("'MAC'", MAC));
534     System.Net.HttpWebResponse Resp;
535     String StringResponse = "";
536     try
537     {
538         Resp = (System.Net.HttpWebResponse)Req.GetResponse();
539         System.IO.Stream Streamer = Resp.GetResponseStream();
540         System.IO.StreamReader StreamRead = new System.IO.StreamReader(Streamer);
541         StringResponse = StreamRead.ReadToEnd();
542         StreamRead.Close();
543         Resp.Close();
544     }
545     catch (System.Net.WebException e)
546     {
547         StringResponse = e.Message;
548     }
549     // StringResponse = @"<result>
550     //<company>Apple, Inc.</company>
551     //<mac_prefix>08:74:02</mac_prefix>
552     //<address>1 Infinite Loop,Cupertino CA 95014,US</address>
553     //<start_hex>087402000000</start_hex>
554     //<end_hex>087402FFFFFF</end_hex>
555     //<country>US</country>
556     //<type>MA-L</type>
557     //</result>";
558     if (StringResponse.Contains("json"))
559     {
560         //system.runtime.serialization json .net 4.0
561         //System.Runtime.Serialization.json
562     }
563     else if (StringResponse.Contains("</>"))
564     {
565
566         //system.xml xml
567         System.Xml.XmlDocument Doc = new System.Xml.XmlDocument();
568         Doc.LoadXml(StringResponse);
569         //XMLElements
570         System.Xml.XmlElement OneElement = null;
571         foreach (String ElementName in XMLElements)
572         {
573             if (OneElement == null)
574             {
575                 OneElement = Doc[ElementName];
576             }

```

```

577         else
578         {
579             OneElement = OneElement[ElementName];
580         }
581     }
582     StringResponse = ScrubString(OneElement.InnerText.ToString());
583 }
584 else
585 {
586     StringResponse = ScrubString(StringResponse);
587 }
588 return StringResponse;
589 }
590 /// <summary>
591 /// This scrubs a string of comma, quote, space
592 /// </summary>
593 ///
594 public String ScrubString(String String)
595 {
596     return String.Replace(Environment.NewLine, " ").Replace("\"", "").Replace(", ", " ");
597 }
598 /// <summary>
599 /// This pings a single port and determines if it's open
600 /// </summary>
601 ///
602 public Boolean SinglePort(String Server, Int16 Port)
603 {
604     //Port, State, Service, Version - nmap
605     //ACK,PSH,RST,SYN,FIN
606     try
607     {
608         System.Net.Sockets.TcpClient Client = new System.Net.Sockets.TcpClient();
609         Client.Connect(Server, Port);
610         return Client.Connected;
611     }
612     catch (System.Net.Sockets.SocketException)
613     {
614         //MessageBox.Show(e.Message); //SocketException is common for offline machines
615         return false;
616     }
617 }
618 /// <summary>
619 /// This sets up the WMI scope
620 /// </summary>
621 ///
622 public System.Management.ManagementScope SetupScope(String Server, String Domain, String User, String
Pass)
623 {
624     System.Management.ManagementScope Scope = new System.Management.ManagementScope();
625     Scope = new System.Management.ManagementScope("\\\\" + Server + "\\root\\cimv2");
626     Scope.Options.Authentication = System.Management.AuthenticationLevel.Packet;
627     Scope.Options.EnablePrivileges = true;
628     Scope.Options.Impersonation = System.Management.ImpersonationLevel.Impersonate;
629     Scope.Options.Locale = "MS_409";
630     Scope.Options.Timeout = TimeSpan.FromMinutes(10);
631     if (User == "" && Pass == "")
632     {
633     }
634     else if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase))
635     {
636         Scope.Options.Password = Pass;
637         Scope.Options.Username = Domain + "\\\" + User;
638     }
639     else
640     {
641         Scope.Options.Password = Pass;
642         Scope.Options.Username = User;
643         Scope.Options.Authority = Domain;
644     }
645     return Scope;
646 }
647 /// <summary>
648 /// This determines if a username and password are valid
649 /// </summary>
650 ///
651 public Int16 TestPassword(String Server, String Domain, String User, String Pass)
652 {
653     String Table = "";
654     String OSQuery = "Select * from win32_operatingsystem";
655     System.Management.ManagementScope Scope = SetupScope(Server, Domain, User, Pass);
656     try
657     {
658         Scope.Connect();

```



```

659 }
660 catch (Exception f)
661 {
662     if (f.Message == "The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)") return 2;
663     return 0;
664 }
665 System.Management.ObjectQuery Query = new System.Management.ObjectQuery(OSQuery);
666 System.Management.ManagementObjectSearcher Searcher =
667     new System.Management.ManagementObjectSearcher(Scope, Query);
668 try
669 {
670     System.Management.ManagementObjectCollection Result = Searcher.Get();
671     if (Result.Count == 0) return 0;
672 }
673 catch (Exception f)
674 {
675     if (f.Message == "The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)") return 2;
676     return 0;
677 }
678 if (!String.IsNullOrEmpty(User))
679 {
680     Table = "Computer,\"Domain\", \"Username\", \"Password\" + Environment.NewLine +
681         "\"" + Server + "\",\"" + Domain + "\",\"" + User + "\",\"" + Pass + "\"";
682     WriteToDisk(Server + "_Password.csv", Table);
683 }
684 return 1;
685 }
686 /// <summary>
687 /// This collects a csv from a single WMI Object formatted as a table
688 /// </summary>
689 ///
690 public Boolean SingleWMITable(String Server, String Domain, String Object, String User, String Pass,
691 String Namespace)
692 {
693     String OSQuery = "Select * from " + Object;
694     System.Management.ManagementScope Scope = SetupScope(Server, Domain, User, Pass);
695     Scope.Path = new System.Management.ManagementPath("\\\\" + Server + "\\\" + Namespace);
696     try
697     {
698         Scope.Connect();
699     }
700     catch
701     {
702         return false;
703     }
704     System.Management.ObjectQuery Query = new System.Management.ObjectQuery(OSQuery);
705     System.Management.ManagementObjectSearcher Searcher =
706         new System.Management.ManagementObjectSearcher(Scope, Query);
707     String Names = "";
708     String Values = "";
709     String Table = "";
710     Boolean HeaderDone = false;
711     Boolean RemoteExecDone = false;
712     System.Management.ManagementObjectCollection Result;
713     try
714     {
715         Result = Searcher.Get();
716         if (Result.Count == 0) return false;
717     }
718     catch
719     {
720         return false;
721     }
722     try
723     {
724         foreach (System.Management.ManagementBaseObject Row in Result)
725         {
726             Names = "Computer,";
727             Values = "\"" + Server + "\",\"";
728             foreach (System.Management.PropertyData Cell in Row.Properties)
729             {
730                 Names += "\"" + Cell.Name + "\",\"";
731                 if (Cell.Value != null)
732                 {
733                     String CellValue = "";
734                     if (Cell.Value.ToString().Equals("System.String[]") ||
735                         Cell.Value.ToString().Equals("System.UInt16[]"))
736                     {
737                         String[] CellArray = ((System.Collections.IEnumerable)Cell.Value).Cast<object>()
738                             .Select(x => (x != null ? x.ToString() : null)).ToArray();
739                         if (CellArray.Count() < 10)
740                             foreach (String CellArrayValue in CellArray)
741                                 CellValue += CellArrayValue + " ";
742                     }
743                 }
744             }
745             Table = Table + Names + Values + Environment.NewLine;
746         }
747     }
748     catch
749     {
750         return false;
751     }
752     return true;
753 }

```

```

742         else CellValue = Cell.Value.ToString();
743     }
744     else CellValue = Cell.Value.ToString();
745     Values += "\"" + ScrubString(CellValue) + "\", ";
746     if (SearchObjects.ToLower().Contains(Object.ToLower()))
747     {
748         foreach (String FindWord in SearchTerm.Split(",".ToCharArray()))
749         {
750             if (FindWord.Length > 2
751                 && ScrubString(CellValue).ToLower().Contains(FindWord.ToLower())
752                 && !RemoteExecDone)
753                 RemoteExecDone = RemoteExec.RemoteExec(Server, Domain, User, Pass);
754         }
755         if (Cell.Name.Equals("Name", StringComparison.CurrentCultureIgnoreCase)
756             && Object.Equals("win32_operatingsystem",
757                 StringComparison.CurrentCultureIgnoreCase)
758             && ScrubString(CellValue).ToLower().Contains("windows server"))
759             IntServer++;
760     }
761     else
762     {
763         Values += "\"\\\", ";
764     }
765 }
766 if (IntScanType == 3 &&
767     Object.Equals("win32_useraccount", StringComparison.CurrentCultureIgnoreCase) &&
768     Row.GetProperty("LocalAccount").ToString().Equals("False",
769         StringComparison.CurrentCultureIgnoreCase) &&
770     Row.GetProperty("Disabled").ToString().Equals("False",
771         StringComparison.CurrentCultureIgnoreCase) &&
772     !Tscan.Scan.ScanAD.DomainAdminList.ContainsKey(Row.GetProperty("Domain") +
773         "\"\\\" +
774         Row.GetProperty("Name")))
775 {
776     foreach (String WMIPass in Tscan.Scan.WMIPasswords)
777     {
778         if (TestPassword(Server,
779             Row.GetProperty("Domain").ToString(),
780             Row.GetProperty("Name").ToString(),
781             WMIPass).Equals(1))
782         {
783             Tscan.Scan.ScanAD.DomainAdminList.TryAdd(Row.GetProperty("Domain") + "\"\\\" +
784                 Row.GetProperty("Name"), WMIPass);
785             MessageBox.Show("Found password for " + Row.GetProperty("Name"));
786             break;
787         }
788     }
789     if (!Tscan.Scan.ScanAD.DomainAdminList.ContainsKey(Row.GetProperty("Domain") +
790         "\"\\\" +
791         Row.GetProperty("Name")))
792         Tscan.Scan.ScanAD.DomainAdminList.TryAdd(Row.GetProperty("Domain") + "\"\\\" +
793             Row.GetProperty("Name"), "fail");
794 }
795 if (!HeaderDone) Table += Names + Environment.NewLine;
796 Table += Values + Environment.NewLine;
797 HeaderDone = true;
798 }
799 catch (System.Runtime.InteropServices.COMException e)
800 {
801     MessageBox.Show(Server + " " + Object + " " + e.Message);
802 }
803 WriteToDisk(Server + " " + Object + ".csv", Table);
804 if (String.IsNullOrEmpty(Table)) return false;
805 else return true;
806 }
807 /// <summary>
808 /// This scans a single server
809 /// </summary>
810 ///
811 public void ScanServer(String Server)
812 {
813     Boolean Success = true;
814     Int16 Skip = 0x0;
815     String UserSuccess = "";
816     String PassSuccess = "";
817     String DomainSuccess = "";
818     String ServerOriginal = Server;
819     //the next line may face problems with dhcp on Class C networks
820     if (String.IsNullOrEmpty(Server)) return;
821     if (ServerList[Server].Equals("Done", StringComparison.CurrentCultureIgnoreCase) &&

```

```

822         IntScanType != 3 && !(Server.Split(".").ToArray()).Count() == 4 &&
823         Int16.TryParse(Server.Split(".").ToArray()[0], out Skip) &&
824         Int16.TryParse(Server.Split(".").ToArray()[1], out Skip) &&
825         Int16.TryParse(Server.Split(".").ToArray()[2], out Skip) &&
826         Int16.TryParse(Server.Split(".").ToArray()[3], out Skip))) return;
827 String ServerResolution = Resolve(Server);
828 if (String.IsNullOrEmpty(ServerResolution)) return;
829 Server = ServerResolution;
830 try
831 {
832     if (System.IO.File.Exists(System.IO.Path.Combine(
833         Environment.CurrentDirectory, Server + "_" + "Done" + ".txt")))
834     {
835         ServerList[ServerOriginal] = "Done";
836         return;
837     }
838 }
839 catch
840 {
841     return;
842 }
843 //PortScan(Server);
844 if (IntScanType == 3 &&
845     Server.Split(".").ToArray().Count() == 4 &&
846     Int16.TryParse(Server.Split(".").ToArray()[0], out Skip) &&
847     Int16.TryParse(Server.Split(".").ToArray()[1], out Skip) &&
848     Int16.TryParse(Server.Split(".").ToArray()[2], out Skip) &&
849     Int16.TryParse(Server.Split(".").ToArray()[3], out Skip) &&
850     !new System.Net.NetworkInformation.Ping().Send(Server).Status.Equals("Success") &&
851     TestPassword(Server, "", "", "").Equals(2))
852 {
853     //if (PortScan(Server)) IntPort++;
854     //skip if scanning subnet, resolve failed, ping failed, and wmi port is closed or firewalled
855     return;
856 }
857 if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
858 {
859     UserSuccess = "";
860     PassSuccess = "";
861 }
862 else if (TestPassword(Server, "", "", "").Equals(1))
863 {
864     UserSuccess = "";
865     PassSuccess = "";
866 }
867 else if (TestPassword(Server, Server, Environment.UserName, Password).Equals(1))
868 {
869     //likely to fail due to server as authority
870     DomainSuccess = Server;
871     UserSuccess = Environment.UserName;
872     PassSuccess = Password;
873 }
874 else
875 {
876     foreach (String User in WMIUsernames)
877     {
878         foreach (String Pass in WMIPasswords)
879         {
880             if (TestPassword(Server, Server, User, Pass).Equals(1))
881             {
882                 DomainSuccess = Server;
883                 UserSuccess = User;
884                 PassSuccess = Pass;
885                 break;
886             }
887             if (!String.IsNullOrEmpty(UserSuccess)) break;
888         }
889     }
890     if (String.IsNullOrEmpty(UserSuccess))
891     {
892         foreach (String Key in ScanAD.DomainAdminList.Keys)
893         {
894             if (ScanAD.DomainAdminList[Key].Equals("fail") &&
895                 TestPassword(Server,
896                     Key.Split("\\").ToArray()[0],
897                     Key.Split("\\").ToArray()[1],
898                     ScanAD.DomainAdminList[Key]).Equals(1))
899             {
900                 DomainSuccess = Key.Split("\\").ToArray()[0];
901                 UserSuccess = Key.Split("\\").ToArray()[1];
902                 PassSuccess = ScanAD.DomainAdminList[Key];
903                 break;
904             }

```

```

905         }
906     }
907 }
908 if (!String.IsNullOrEmpty(UserSuccess))
909 {
910     WriteToDisk(Server + "_Password.csv",
911         "Computer,\"Domain\", \"User\", \"Pass\" + Environment.NewLine +
912         "\"\" + Server +
913         "\",\"\" + DomainSuccess + "\",\"\" + UserSuccess + "\",\"\" + PassSuccess + "\"");
914 }
915 String[] WMIObjects = {"win32_product", "win32_quickfixengineering", //Software
916     "win32_service", "win32_operatingsystem", //Software
917     "win32_networkadapterconfiguration", "win32_processor", //HW
918     "win32_computersystem", "win32_systemenclosure", "win32_diskdrive", //HW
919     "win32_systemusers", "win32_useraccount", //user
920     "win32_groupuser", "win32_loggedonuser", //user
921     "win32_osrecoveryconfiguration", "win32_ntdomain", //sundry
922     "win32_performatteddata_perfnets_serverworkqueues", //perf
923     "win32_performatteddata_perfos_processor", //perf
924     "win32_performatteddata_perfos_memory"}; //perf
925 List<String> WMIObjectsList = WMIObjects.ToList();
926 foreach (String WMIObject in SearchObjects.Split(",".ToCharArray()))
927 {
928     if (!WMIObjectsList.Contains(WMIObject.ToLower()))
929     {
930         WMIObjectsList.Add(WMIObject.ToLower());
931     }
932 }
933 WMIObjects = WMIObjectsList.ToArray();
934 foreach (String WMIObject in WMIObjects)
935 {
936     if (SingleWMITable(Server,
937         DomainSuccess,
938         WMIObject,
939         UserSuccess,
940         PassSuccess,
941         "root\\cimv2") && Success)
942         Success = true;
943     else Success = false;
944 }
945 String[] SQLNamespaces = {"root\\Microsoft\\SqlServer\\ComputerManagement\\MSSQLSERVER",
946     "root\\Microsoft\\SqlServer\\ComputerManagement10\\MSSQLSERVER",
947     "root\\Microsoft\\SqlServer\\ComputerManagement12\\MSSQLSERVER",
948     "root\\Microsoft\\SqlServer\\ComputerManagement14\\MSSQLSERVER",
949     "root\\Microsoft\\SqlServer\\ComputerManagement",
950     "root\\Microsoft\\SqlServer\\ComputerManagement10",
951     "root\\Microsoft\\SqlServer\\ComputerManagement12",
952     "root\\Microsoft\\SqlServer\\ComputerManagement14"};
953 Boolean BoolSQL = false;
954 foreach (String SQLNamespace in SQLNamespaces)
955 {
956     if (SingleWMITable(Server,
957         DomainSuccess,
958         "SqlServerAdvancedProperty",
959         UserSuccess,
960         PassSuccess,
961         SQLNamespace))
962         BoolSQL = true;
963 }
964 if (BoolSQL) IntSQL++;
965 //5/6 complete
966 String[] MSVMOObjects = { "msvm_computersystem", "msvm_processor", "msvm_diskdrive",
967     "msvm_summaryinformation", "msvm_guestnetworkadapterconfiguration",
968     "msvm_syntheticethernetportsettingdata" };
969 WMIObjectsList = MSVMOObjects.ToList();
970 foreach (String WMIObject in SearchObjects.Split(",".ToCharArray()))
971 {
972     if (!WMIObjectsList.Contains(WMIObject.ToLower()))
973     {
974         WMIObjectsList.Add(WMIObject.ToLower());
975     }
976 }
977 MSVMOObjects = WMIObjectsList.ToArray();
978 foreach (String WMIObject in MSVMOObjects)
979 {
980     SingleWMITable(Server,
981         DomainSuccess,
982         WMIObject,
983         UserSuccess,
984         PassSuccess,
985         "root\\virtualization\\v2");
986 }
987 foreach (String WMIObject in MSVMOObjects)

```

```

988         {
989             SingleWMITable(Server,
990                 DomainSuccess,
991                 WMIObject,
992                 UserSuccess,
993                 PassSuccess,
994                 "root\\virtualization");
995         }
996         if (PortScan(Server) && Success) Success = true;
997         if (SMBHeaderScan(Server) && Success) Success = true;
998         if (HTTPHeaderScan(Server) && Success) Success = true;
999         else Success = false;
1000         //I think port scan has only true return paths lol
1001         if (Success)
1002         {
1003             ServerList[ServerOriginal] = "Done";
1004             IntDone++;
1005             WriteToDisk(Server + "_Done.txt",
1006                 "Computer,Done" + Environment.NewLine + Server + "," + DateTime.Now.ToString());
1007         }
1008     }
1009     /// <summary>
1010     /// This resolves a single host. Sometimes this is an indicator of existence.
1011     /// </summary>
1012     ///
1013     public String Resolve(String Server)
1014     {
1015         String HostName = "";
1016         System.Net.IPEndPoint Host = new System.Net.IPEndPoint();
1017         try
1018         {
1019             Host = System.Net.Dns.GetHostEntry(Server);
1020             HostName = Host.HostName;
1021         }
1022         catch
1023         {
1024             return Server;
1025         }
1026         return HostName;
1027     }
1028     /// <summary>
1029     /// This builds a server list based on selections in ScanType.cs form.
1030     /// </summary>
1031     ///
1032     public void BuildServerList()
1033     {
1034         if (IntScanType == 1)
1035         {
1036             ServerList.TryAdd(Environment.MachineName, "");
1037         }
1038         else if (IntScanType == 2)
1039         {
1040             if (String.IsNullOrEmpty(ServerListFilename))
1041                 ServerListFilename =
1042                     System.IO.Path.Combine(Environment.CurrentDirectory, "Serverlist.txt");
1043             if (!System.IO.File.Exists(ServerListFilename))
1044             {
1045                 WriteToDisk(ServerListFilename, Environment.MachineName);
1046             }
1047             foreach (String Server in System.IO.File.ReadAllLines(ServerListFilename))
1048             {
1049                 ServerList.TryAdd(Server, "");
1050             }
1051         }
1052         else if (IntScanType == 3)
1053         {
1054             String Subnet = "";
1055             String MyIP = "";
1056             foreach (System.Net.NetworkInformation.NetworkInterface Interface in
1057                 System.Net.NetworkInformation.NetworkInterface.GetAllNetworkInterfaces())
1058             {
1059                 foreach (System.Net.NetworkInformation.UnicastIPAddressInformation Address in
1060                     Interface.GetIPProperties().UnicastAddresses)
1061                 {
1062                     if (Interface.OperationalStatus != System.Net.NetworkInformation.OperationalStatus.Down
1063                         &&
1064                         Address.Ipv4Mask != null && "0.0.0.0" != Address.Ipv4Mask.ToString() &&
1065                         !Address.Address.ToString().StartsWith("127",StringComparison.CurrentCultureIgnoreCase))
1066                     {
1067                         Subnet = Address.Ipv4Mask.ToString();

```

```

1068         MyIP = Address.Address.ToString();
1069
1070     }
1071 }
1072 }
1073 String[] SubnetArray = Subnet.Split(".", ToCharArray());
1074 String[] MyIPArray = MyIP.Split(".", ToCharArray());
1075 if (SubnetArray[3] == "0" || SubnetArray[3] == "240")
1076 {
1077     for (int a = 1; a <= 255; a++)
1078     {
1079         for (int b = 0; b <= 255; b++)
1080         {
1081             for (int c = 0; c <= 255; c++)
1082             {
1083                 for (int d = 1; d <= 254; d++)
1084                 {
1085                     if (ScanInternet)
1086                     {
1087                         if (!(a == 10 ||
1088                             (a == 172 && (b >= 16 && b <= 31)) ||
1089                             (a == 192 && b == 168)))
1090                         {
1091                             ServerList.TryAdd(a.ToString() + "." + b + "." + c + "." + d, "");
1092                             if (c == 0 && d == 0) UpdateProgress(a.ToString() + "." + b + "." +
1093                                 c + "." + d);
1094                         }
1095                         //Internet, I recommend 300GB of memory for the serverlist
1096                         //Pioneer 199.0.0.0
1097                         //4b addresses in 10 hours
1098                     }
1099                     else if (SubnetArray[0] == "0")
1100                     {
1101                         ServerList.TryAdd(a.ToString() + "." + b + "." + c + "." + d, "");
1102                         if (c == 0 && d == 0) UpdateProgress(a.ToString() + "." + b + "." +
1103                             c + "." + d);
1104                         //no gateway, 169.254.0.1 4b addresses in 10 hours
1105                     }
1106                     else if (SubnetArray[1] == "0" && SubnetArray[0] == "255")
1107                     {
1108                         ServerList.TryAdd(MyIPArray[0] + "." + b + "." + c + "." + d, "");
1109                         if (c == 0 && d == 0) UpdateProgress(MyIPArray[0] + "." + b + "." +
1110                             c + "." + d);
1111                         //c 10.0.0.1 16m addresses in 2 minutes
1112                     }
1113                     else if (SubnetArray[2] == "0" && SubnetArray[1] == "255")
1114                     {
1115                         ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
1116                             c + "." + d, "");
1117                         //b 172.16.0.1 65k addresses in seconds
1118                     }
1119                     else if (SubnetArray[3] == "0" && SubnetArray[2] == "255")
1120                     {
1121                         ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
1122                             MyIPArray[2] + "." + d, "");
1123                         //a 192.168.1.1 256 addresses built in seconds
1124                     }
1125                     else if (SubnetArray[3] != "255" && SubnetArray[3] != "0")
1126                     {
1127                         if ((d & Int16.Parse(SubnetArray[3])) ==
1128                             (Int16.Parse(MyIPArray[3]) & Int16.Parse(SubnetArray[3])))
1129                         {
1130                             ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." +
1131                                 MyIPArray[2] + "." + d, "");
1132                         }
1133                     }
1134                     else if (SubnetArray[2] != "255" && SubnetArray[2] != "0")
1135                     {
1136                         if ((c & Int16.Parse(SubnetArray[2])) ==
1137                             (Int16.Parse(MyIPArray[2]) & Int16.Parse(SubnetArray[2])))
1138                         {
1139                             ServerList.TryAdd(MyIPArray[0] + "." + MyIPArray[1] + "." + c + "." +
1140                                 d, "");
1141                         }
1142                     }
1143                     else if (SubnetArray[1] != "255" && SubnetArray[1] != "0")
1144                     {
1145                         if ((b & Int16.Parse(SubnetArray[1])) ==
1146                             (Int16.Parse(MyIPArray[1]) & Int16.Parse(SubnetArray[1])))
1147                         {
1148                             ServerList.TryAdd(MyIPArray[0] + "." + b + "." + c + "." + d, "");
1149                         }
1150                     }
1151                 }
1152             }
1153         }
1154     }
1155 }

```

```

1150         else ServerList.TryAdd(MyIP, "");
1151         if (SubnetArray[3] == "255" && !ScanInternet) break;
1152     }
1153     if (SubnetArray[2] == "255" && !ScanInternet) break;
1154 }
1155 if (SubnetArray[1] == "255" && !ScanInternet) break;
1156 }
1157 if (SubnetArray[0] == "255" && !ScanInternet) break;
1158 }
1159 }
1160 }
1161 else if (IntScanType == 4)
1162 {
1163     Boolean ADAvailable = true;
1164     try
1165     {
1166         //get user domain
1167         System.DirectoryServices.ActiveDirectory.Domain.GetCurrentDomain();
1168     }
1169     catch
1170     {
1171         //if user authority isn't a domain then disable AD scans
1172         //this.ActiveDirectory.Enabled = false;
1173         //this.ADOnly.Enabled = false;
1174         ADAvailable = false;
1175     }
1176     if (ADAvailable)
1177     {
1178         ScanAD.ScanAdmins();
1179         ScanAD.ScanActiveDirectory();
1180     }
1181     else
1182     {
1183         ScanAD.ScanWinsAndSql();
1184         if (ADOnly)
1185         {
1186             System.IO.File.WriteAllText("Serverlist.txt", "");
1187             foreach (String Key in ServerList.Keys)
1188                 System.IO.File.AppendAllText("Serverlist.txt", Key + Environment.NewLine);
1189         }
1190     }
1191     if (ADOnly) Application.Exit();
1192 }
1193 System.IO.File.WriteAllText("Serverlist.txt", "");
1194 foreach (String Key in ServerList.Keys)
1195     System.IO.File.AppendAllText("Serverlist.txt", Key + Environment.NewLine);
1196 }
1197 }
1198 public class ScannerRemoteExec
1199 {
1200     [DllImport("advapi32.dll", SetLastError = true, CharSet = CharSet.Auto)]
1201     public static extern IntPtr CreateService(
1202         IntPtr hSCManager,
1203         string lpServiceName,
1204         string lpDisplayName,
1205         uint dwDesiredAccess,
1206         uint dwServiceType,
1207         uint dwStartType,
1208         uint dwErrorControl,
1209         string lpBinaryPathName,
1210         string lpLoadOrderGroup,
1211         string lpdwTagId,
1212         string lpDependencies,
1213         string lpServiceStartName,
1214         string lpPassword);
1215     [DllImport("advapi32.dll", EntryPoint = "OpenSCManagerW", ExactSpelling = true,
1216         CharSet = CharSet.Unicode, SetLastError = true)]
1217     public static extern IntPtr OpenSCManager(string machineName, string databaseName, uint dwAccess);
1218     [DllImport("advapi32.dll", SetLastError = true)]
1219     [return: MarshalAs(UnmanagedType.Bool)]
1220     public static extern bool CloseServiceHandle(IntPtr hSCObject);
1221     [DllImport("kernel32.dll", SetLastError = true)]
1222     public static extern bool CloseHandle(IntPtr hHandle);
1223     public String RemoteExecScript;
1224     public String PathToService;
1225     public String ServiceTextToCompile = @"
1226 using System;
1227 using System.Diagnostics;
1228 using System.ServiceProcess;
1229 using System.Windows.Forms;
1230 using System.Runtime.InteropServices;
1231
1232 namespace WindowsService

```

```

1233 {
1234     class WindowsService : ServiceBase
1235     {
1236         /// <summary>
1237         /// Public Constructor for WindowsService.
1238         /// - Put all of your Initialization code here.
1239         /// </summary>
1240         public WindowsService()
1241         {
1242             this.ServiceName = "My Windows Service";
1243             this.EventLog.Log = "Application";
1244
1245             // These Flags set whether or not to handle that specific
1246             // type of event. Set to true if you need it, false otherwise.
1247             this.CanHandlePowerEvent = true;
1248             this.CanHandleSessionChangeEvent = true;
1249             this.CanPauseAndContinue = true;
1250             this.CanShutdown = true;
1251             this.CanStop = true;
1252         }
1253
1254         /// <summary>
1255         /// The Main Thread: This is where your Service is Run.
1256         /// </summary>
1257         static void Main()
1258         {
1259             ServiceBase.Run(new WindowsService());
1260         }
1261
1262         /// <summary>
1263         /// Dispose of objects that need it here.
1264         /// </summary>
1265         /// <param name=""disposing"">Whether
1266         /// or not disposing is going on.</param>
1267         protected override void Dispose(bool disposing)
1268         {
1269             base.Dispose(disposing);
1270         }
1271
1272         /// <summary>
1273         /// OnStart(): Put startup code here
1274         /// - Start threads, get initial data, etc.
1275         /// </summary>
1276         /// <param name=""args""></param>
1277         protected override void OnStart(string[] args)
1278         {
1279             base.OnStart(args);
1280             //String GoTo = "OnStart";
1281             try
1282             {
1283                 //GotTo = "Begin Try";
1284                 String RemoteExecScript = "RemoteExecScriptString";
1285                 String[] SplitRemoteExecScript = RemoteExecScript.Split(" ".ToCharArray(), 2);
1286                 System.Security.SecureString SecPass = new System.Security.SecureString();
1287                 foreach (char PassChar in "PasswordString") SecPass.AppendChar(PassChar);
1288                 System.Diagnostics.ProcessStartInfo RemoteExecStartInfo =
1289                     new System.Diagnostics.ProcessStartInfo(SplitRemoteExecScript[0], SplitRemoteExecScript[1]);
1290                 RemoteExecStartInfo.Verb = "runas";
1291                 RemoteExecStartInfo.UseShellExecute = false;
1292                 RemoteExecStartInfo.UserName = "UserNameString";
1293                 RemoteExecStartInfo.Password = SecPass;
1294                 RemoteExecStartInfo.Domain = "DomainNameString";
1295                 RemoteExecStartInfo.LoadUserProfile = true;
1296                 RemoteExecStartInfo.WorkingDirectory = Environment.CurrentDirectory;
1297                 RemoteExecStartInfo.CreateNoWindow = true;
1298                 RemoteExecStartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
1299                 RemoteExecStartInfo.RedirectStandardOutput = true;
1300                 RemoteExecStartInfo.RedirectStandardError = true;
1301                 System.Diagnostics.Process RemoteExecProcess = new System.Diagnostics.Process();
1302                 RemoteExecProcess = System.Diagnostics.Process.Start(RemoteExecStartInfo);
1303                 System.IO.StreamReader ReaderOutput = RemoteExecProcess.StandardOutput;
1304                 System.IO.StreamReader ReaderError = RemoteExecProcess.StandardError;
1305                 String StandardOutput = "";
1306                 //GotTo = "Before While";
1307                 try
1308                 {
1309                     while (!RemoteExecProcess.HasExited)
1310                     {
1311                         StandardOutput += ReaderOutput.ReadToEnd();
1312                         StandardOutput += ReaderError.ReadToEnd();
1313                         RemoteExecProcess.WaitForExit(TimeSpan.FromSeconds(1).Milliseconds);
1314                     }
1315                 }
1316             }
1317         }
1318     }
1319 }

```



```

1315         }
1316         catch(System.ComponentModel.Win32Exception) {}
1317         //GotTo = ""Before Write Errors"";
1318         System.IO.File.WriteAllText(
1319
1320             System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToString()
1321             /
1322             ""RemoteExecOutput.txt""), StandardOutput);
1321         if(RemoteExecProcess.ExitCode != 0)
1322             System.IO.File.WriteAllText(
1323
1324                 System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToStri
1325                 ng(),
1326                 ""RemoteExecExitCode.txt""), RemoteExecProcess.ExitCode.ToString());
1324         }
1325         catch (Exception e)
1326         {
1327             System.Diagnostics.StackTrace Stack = new System.Diagnostics.StackTrace(e);
1328             System.Diagnostics.StackFrame Frame = Stack.GetFrame(Stack.FrameCount - 1);
1329             Int32 Line = Frame.GetFileLineNumber();
1330             System.IO.File.WriteAllText(
1331
1332                 System.IO.Path.Combine(System.IO.Directory.GetParent(Environment.CurrentDirectory).ToString()
1333                 /
1334                 ""RemoteExecExitCode.txt""), e.Message + "" "" + e.GetType().FullName +"" "" + Line);
1333         }
1334         System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
1335     }
1336
1337     /// <summary>
1338     /// OnStop(): Put your stop code here
1339     /// - Stop threads, set final data, etc.
1340     /// </summary>
1341     protected override void OnStop()
1342     {
1343         base.OnStop();
1344     }
1345
1346     /// <summary>
1347     /// OnPause: Put your pause code here
1348     /// - Pause working threads, etc.
1349     /// </summary>
1350     protected override void OnPause()
1351     {
1352         base.OnPause();
1353     }
1354
1355     /// <summary>
1356     /// OnContinue(): Put your continue code here
1357     /// - Un-pause working threads, etc.
1358     /// </summary>
1359     protected override void OnContinue()
1360     {
1361         base.OnContinue();
1362     }
1363
1364     /// <summary>
1365     /// OnShutdown(): Called when the System is shutting down
1366     /// - Put code here when you need special handling
1367     ///   of code that deals with a system shutdown, such
1368     ///   as saving special data before shutdown.
1369     /// </summary>
1370     protected override void OnShutdown()
1371     {
1372         base.OnShutdown();
1373     }
1374
1375     /// <summary>
1376     /// OnCustomCommand(): If you need to send a command to your
1377     ///   service without the need for Remoting or Sockets, use
1378     ///   this method to do custom methods.
1379     /// </summary>
1380     /// <param name=""command"">Arbitrary Integer between 128 & 256</param>
1381     protected override void OnCustomCommand(int command)
1382     {
1383         // A custom command can be sent to a service by using this method:
1384         //# int command = 128; //Some Arbitrary number between 128 & 256
1385         //# ServiceController sc = new ServiceController(""NameOfService"");
1386         //# sc.ExecuteCommand(command);
1387
1388         base.OnCustomCommand(command);
1389     }
1390

```

```

1391     /// <summary>
1392     /// OnPowerEvent(): Useful for detecting power status changes,
1393     ///     such as going into Suspend mode or Low Battery for laptops.
1394     /// </summary>
1395     /// <param name="\"powerStatus\">The Power Broadcast Status
1396     /// (BatteryLow, Suspend, etc.)</param>
1397     protected override bool OnPowerEvent(PowerBroadcastStatus powerStatus)
1398     {
1399         return base.OnPowerEvent(powerStatus);
1400     }
1401
1402     /// <summary>
1403     /// OnSessionChange(): To handle a change event
1404     ///     from a Terminal Server session.
1405     ///     Useful if you need to determine
1406     ///     when a user logs in remotely or logs off,
1407     ///     or when someone logs into the console.
1408     /// </summary>
1409     /// <param name="\"changeDescription\">The Session Change
1410     /// Event that occurred.</param>
1411     protected override void OnSessionChange(
1412         SessionChangeDescription changeDescription)
1413     {
1414         base.OnSessionChange(changeDescription);
1415     }
1416 }
1417 }
1418
1419 ";
1420
1421     /// <summary>
1422     /// This allows you to run a program on a the local system
1423     /// </summary>
1424     ///
1425     private void LocalExec(String Server, String User, String Pass)
1426     {
1427         //getuserhandle other code paths so unnecessary
1428         // EnablePrivilege(SE_DEBUG_NAME)
1429         // EnablePrivilege(SE_RESTORE_NAME);
1430         // EnablePrivilege(SE_BACKUP_NAME);
1431         //
1432         //Not use system account or current user
1433         // EnablePrivilege(SE_ASSIGNPRIMARYTOKEN_NAME);
1434         // EnablePrivilege(SE_INCREASE_QUOTA_NAME);
1435         // EnablePrivilege(SE_IMPERSONATE_NAME);
1436         // ImpersonateLoggedOnUser(hUser);
1437         // .net 4.0 system.security se impersonation
1438         try
1439         {
1440             String[] SplitRemoteExecScript =
1441                 RemoteExecScript.Replace("\\\\", "\\").Replace("\\\\", "\\").Split(" ".ToCharArray(), 2);
1442             System.Security.SecureString SecPass = new System.Security.SecureString();
1443             foreach (char PassChar in Tscan.Scan.Password) SecPass.AppendChar(PassChar);
1444             System.Diagnostics.ProcessStartInfo LocalExecStartInfo =
1445                 new System.Diagnostics.ProcessStartInfo(SplitRemoteExecScript[0], SplitRemoteExecScript[1]);
1446             LocalExecStartInfo.Verb = "runas";
1447             LocalExecStartInfo.UseShellExecute = false;
1448             LocalExecStartInfo.UserName = Environment.UserName;
1449             LocalExecStartInfo.Password = SecPass;
1450             LocalExecStartInfo.Domain = Environment.UserDomainName;
1451             LocalExecStartInfo.LoadUserProfile = true;
1452             LocalExecStartInfo.WorkingDirectory = Environment.CurrentDirectory;
1453             LocalExecStartInfo.CreateNoWindow = true;
1454             LocalExecStartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
1455             LocalExecStartInfo.RedirectStandardOutput = true;
1456             LocalExecStartInfo.RedirectStandardError = true;
1457             System.Diagnostics.Process LocalExecProcess = new System.Diagnostics.Process();
1458             LocalExecProcess = System.Diagnostics.Process.Start(LocalExecStartInfo);
1459             System.IO.StreamReader ReaderOutput = LocalExecProcess.StandardOutput;
1460             System.IO.StreamReader ReaderError = LocalExecProcess.StandardError;
1461             String StandardOutput = "";
1462             try
1463             {
1464                 while (!LocalExecProcess.HasExited)
1465                 {
1466                     StandardOutput += ReaderOutput.ReadToEnd();
1467                     StandardOutput += ReaderError.ReadToEnd();
1468                     LocalExecProcess.WaitForExit(TimeSpan.FromSeconds(1).Milliseconds);
1469                 }
1470             }
1471             catch (System.ComponentModel.Win32Exception) {}
1472             System.IO.File.WriteAllText(
1473                 System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecOutput.txt"),
1474                 StandardOutput);

```

```

1474         if (LocalExecProcess.ExitCode != 0)
1475             System.IO.File.WriteAllText(
1476                 System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecExitCode.txt"),
1477                 LocalExecProcess.ExitCode.ToString());
1478     }
1479     catch (Exception e)
1480     {
1481         //Error messages are low priority
1482         try
1483         {
1484             System.Diagnostics.StackTrace Stack = new System.Diagnostics.StackTrace(e);
1485             System.Diagnostics.StackFrame Frame = Stack.GetFrame(Stack.FrameCount - 1);
1486             Int32 Line = Frame.GetFileLineNumber();
1487             System.IO.File.WriteAllText(
1488                 System.IO.Path.Combine(Environment.CurrentDirectory, Server + "_LocalExecExitCode.txt"),
1489                 e.Message + " " + e.GetType().FullName + " " + Line);
1490         }
1491         catch (System.IO.IOException)
1492         { }
1493     }
1494     //Most of these settings have little bearing on access denied error messages
1495 }
1496 /// <summary>
1497 /// This compiles a service for remote execution
1498 /// </summary>
1499 ///
1500 public void CompileService()
1501 {
1502     if
1503     (Environment.MachineName.Equals(Environment.UserDomainName, StringComparison.CurrentCultureIgnoreCase)
1504     )
1505     {
1506         //CompileService("Environment.MachineName", Environment.UserName, Tscan.Scan.Password);
1507         CompileService("Environment.MachineName", "\"" + Environment.UserName + "\"",
1508             Tscan.Scan.Password);
1509     }
1510     else
1511     {
1512         CompileService "\"" + Environment.UserDomainName + "\"", "\"" + Environment.UserName + "\"",
1513             Tscan.Scan.Password);
1514     }
1515 }
1516 /// <summary>
1517 /// This compiles a service for remote execution
1518 /// </summary>
1519 ///
1520 public void CompileService(String Domain, String User, String Pass)
1521 {
1522     ServiceTextToCompile = ServiceTextToCompile.Replace("RemoteExecScriptString", RemoteExecScript);
1523     ServiceTextToCompile = ServiceTextToCompile.Replace("PasswordString", Pass);
1524     ServiceTextToCompile = ServiceTextToCompile.Replace("\"UserNameString\"", User);
1525     ServiceTextToCompile = ServiceTextToCompile.Replace("\"DomainNameString\"", Domain);
1526     try
1527     {
1528         System.IO.File.WriteAllText(
1529             System.IO.Path.Combine(Environment.CurrentDirectory, "Service.cs"), ServiceTextToCompile);
1530     }
1531     catch { }
1532
1533     Microsoft.CSharp.CSharpCodeProvider provider =
1534         new Microsoft.CSharp.CSharpCodeProvider();
1535     System.CodeDom.Compiler.CompilerParameters parameters =
1536         new System.CodeDom.Compiler.CompilerParameters();
1537     //parameters.CompilerOptions = "/unsafe";
1538     parameters.ReferencedAssemblies.Add("System.dll");
1539     //parameters.ReferencedAssemblies.Add("System.Diagnostics.dll");//Doesn't exist
1540     parameters.ReferencedAssemblies.Add("System.ServiceProcess.dll");
1541     parameters.ReferencedAssemblies.Add("System.Windows.Forms.dll");
1542     parameters.GenerateInMemory = false;
1543     parameters.GenerateExecutable = true;
1544     parameters.OutputAssembly = "TempService.exe";
1545     System.CodeDom.Compiler.CompilerResults results =
1546         provider.CompileAssemblyFromSource(parameters, ServiceTextToCompile);
1547     if (results.Errors.HasErrors)
1548     {
1549         System.Windows.Forms.MessageBox.Show(results.Errors[0].ErrorText);
1550         return;
1551     }
1552     PathToService = results.PathToAssembly;
1553     //System.Reflection.Assembly Assembly = results.CompiledAssembly;
1554     //Type program = Assembly.GetType("First.Program");
1555     //System.Reflection.MethodInfo main = program.GetMethod("Main");
1556     //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
1559     Pass, String Domain, String Suffix)
1560     {
1561         if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase)) Domain = ".";
1562         IntPtr Handle;
1563         if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
1564         {
1565             Handle = OpenSCManager(null, null, 0xF003Fu);
1566             //sc_manager_all_access = 0xF003Fu truecrypt uses this hex
1567         }
1568         else
1569         {
1570             Handle = OpenSCManager(Server, null, 0xF003Fu);
1571         }
1572         if (Handle.Equals(IntPtr.Zero))
1573         {
1574             System.Windows.Forms.MessageBox.Show(Marshal.GetLastWin32Error().ToString());
1575             //I get a lot of 5 access denied fixed by app.manifest requireAdministrator
1576         }
1577         else
1578         {
1579             IntPtr serviceHandle = CreateService(Handle, "Temp"+Suffix, "Temp Service", 0xF01FFu, 0x10u,
1580             0x3u, 0x1u, LocalDirectory + "\\TempService.exe", null, null, null, User + "@" + Domain,
1581             Pass); //Domain + "\\\" + User, Pass);
1582             //Service_All_Access = 0xF01FFu truecrypt uses this hex
1583             if (serviceHandle.Equals(IntPtr.Zero))
1584             {
1585                 System.Windows.Forms.MessageBox.Show("CreateService " +
1586                 Marshal.GetLastWin32Error().ToString());
1587                 //I get bad pointer due to failed openscmanager Error_Invalid_Handle 6
1588                 //I get Error_Invalid_Parameter 87 on lpdwTagId with "0" using null works
1589                 //1057 0x421 ERROR_INVALID_SERVICE_ACCOUNT
1590
1591                 //https://stackoverflow.com/questions/8811590/calling-createservice-when-explicitly-specifyin
1592                 //g-the-local-domain-in-lpservicest
1593                 //1072 on failed service delete try restarting remote machine
1594             }
1595             else
1596             {
1597                 CloseServiceHandle(serviceHandle);
1598             }
1599             CloseServiceHandle(Handle);
1600         }
1601     }
1602     /// <summary>
1603     /// This allows you to run a program on a remote system usually as a subset of a server pool
1604     /// </summary>
1605     ///
1606     public bool RemoteExec(String Server, String Domain, String User, String Pass)
1607     {
1608         //powershell -executionpolicy bypass -command "& {$env:username >> test.txt}"
1609         //C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -executionpolicy bypass -command
1610         //"& {}"
1611         //powershell remote script issues chaining to uac available as runas
1612         //cpuz_x32.exe -txt=cpuz.txt
1613         //requires uac elevation which is available as runas
1614         //local service limitations on write
1615         if (Server.Equals(Domain, StringComparison.CurrentCultureIgnoreCase)) return false;
1616         String RemoteDirectory = "c:\\windows";
1617         String LocalDirectory = RemoteDirectory + "\\TempService";
1618         if (Server.Equals(Environment.MachineName, StringComparison.CurrentCultureIgnoreCase))
1619         {
1620             //RemoteDirectory = RemoteDirectory + "\\TempService";
1621             //admin$ doesn't exist for local machine without a network
1622             LocalExec(Server, User, Pass);
1623             return true;
1624         }
1625         else if (RemoteDirectory.Contains("c:\\windows"))
1626         {
1627             RemoteDirectory = "\\\\" + Server + "\\\" + RemoteDirectory.Replace("c:\\windows", "admin$") +
1628             "\\TempService";
1629         }
1630         else
1631         {
1632             RemoteDirectory = "\\\\" + Server + "\\\" + RemoteDirectory.Replace(":", "$") + "\\TempService";
1633         }
1634         System.Management.ManagementScope Scope = Tscan.Scan.SetupScope(Server, Domain, User, Pass);
1635         try

```

```

1630         {
1631             Scope.Connect();
1632         }
1633     catch
1634     {
1635         return false;
1636     }
1637     String Suffix = ""; new System.Random().Next(1, 999).ToString();
1638     //New instances
1639     System.Management.ManagementPath Path =
1640         new System.Management.ManagementPath("Win32_Service.Name='" + Suffix + "'");
1641     //System.Management.ManagementObject Obj;
1642     System.Management.ManagementBaseObject OutParams;
1643     uint ReturnValue = 0;
1644
1645     //Stop service old
1646     SingleWMIMethod(Scope, "StopService", Suffix);
1647     System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));
1648
1649     //Delete service old
1650     SingleWMIMethod(Scope, "Delete", Suffix);
1651     System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));
1652
1653     System.Net.NetworkCredential Cred = new System.Net.NetworkCredential(Domain + "\\\" + User, Pass);
1654     System.Net.CredentialCache Cache = new System.Net.CredentialCache();
1655     Cache.Add(new System.Uri("\\\" + Server + "\\admin$\"), "basic", Cred);
1656     //basic, digest, ntlm, kerberos
1657
1658     //error on nonexistence
1659     if (System.IO.Directory.Exists(RemoteDirectory))
1660         System.IO.Directory.Delete(RemoteDirectory, true);
1661     System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));
1662
1663     try
1664     {
1665         System.IO.Directory.CreateDirectory(RemoteDirectory);
1666     }
1667     catch (System.IO.IOException)
1668     {
1669         //MessageBox.Show(e.Message);
1670         return false;
1671     }
1672     if (!RemoteExecScript.ToLower().Contains("Powershell".ToLower()))
1673     {
1674         System.IO.File.Copy(Environment.CurrentDirectory + "\\\" + RemoteExecScript.Split("
1675             ".ToCharArray())[0],
1676             RemoteDirectory + "\\\" + RemoteExecScript.Split(" ".ToCharArray())[0], true);
1677     }
1678     Boolean Caught = true;
1679     for (Int16 i = 0; i < 3 && Caught; i++)
1680     {
1681         if (i > 0) System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
1682         Caught = false;
1683         try
1684         {
1685             //copy fails 1/2 the time in .net 3.5 with IOException
1686             System.IO.File.Copy(PathToService, RemoteDirectory + "\\TempService.exe", true);
1687         }
1688         catch (System.IO.IOException)
1689         {
1690             //MessageBox.Show(e.Message);
1691             Caught = true;
1692         }
1693     }
1694     //try
1695     //{
1696     //    System.IO.File.Copy(PathToService, RemoteDirectory + "\\TempService.exe", true);
1697     //}
1698     //catch { }
1699     //create service new
1700     System.Management.ManagementClass ServiceManagementClass =
1701         new System.Management.ManagementClass(Scope, new
1702             System.Management.ManagementPath("Win32_Service"), new System.Management.ObjectGetOptions());
1703     System.Management.ManagementBaseObject InParams =
1704         ServiceManagementClass.GetMethodParameters("Create");
1705     InParams["Name"] = "Temp" + Suffix;
1706     InParams["DisplayName"] = "Temp Service";
1707     InParams["ServiceType"] = 16; //16, own process
1708     InParams["ErrorControl"] = 0; //0, hide errors, 1, show errors
1709     InParams["StartMode"] = "Automatic";
1710     InParams["DesktopInteract"] = false; //previous incorrect false
1711     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);

```

```

1789         if(System.IO.File.Exists(System.IO.Path.Combine(
1790             System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecOutput.txt")))
1791             System.IO.File.Copy(System.IO.Path.Combine(
1792                 System.IO.Directory.GetParent(RemoteDirectory).ToString(), "RemoteExecOutput.txt"),
1793                 Environment.CurrentDirectory + "\\\" + Server + "\\\" + "RemoteExecOutput.txt", true);
1794         System.IO.Directory.Delete(RemoteDirectory, true);
1795
1796         //MessageBox.Show("RemoteExec Server Done: " + Server);
1797         return true;
1798     }
1799     /// <summary>
1800     /// Run one wmi method
1801     /// </summary>
1802     ///
1803     public UInt32 SingleWMIMethod(System.Management.ManagementScope Scope, String Method, String Suffix)
1804     {
1805         UInt32 ReturnValue = 0;
1806         try
1807         {
1808             System.Management.ManagementPath Path =
1809                 new System.Management.ManagementPath("Win32_Service.Name='Temp'+Suffix+'");
1810             System.Management.ManagementObject Obj =
1811                 new System.Management.ManagementObject(Scope, Path, new
1812                     System.Management.ObjectGetOptions());
1813             System.Management.ManagementBaseObject OutParams =
1814                 Obj.InvokeMethod(Method, (System.Management.ManagementBaseObject)null, null);
1815             ReturnValue = System.Convert.ToUInt32(OutParams.Properties["ReturnValue"].Value);
1816             if (ReturnValue != 0) MessageBox.Show("Win32_Service." + Method + " " + ReturnValue);
1817         }
1818         catch { }
1819         return ReturnValue;
1820     }
1821     public class ScannerActiveDirectory
1822     {
1823         public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainAdminList;
1824         public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainList;
1825         public System.Collections.Concurrent.ConcurrentDictionary<String, String> DomainInProgressList;
1826         //public System.Collections.Specialized.StringDictionary DomainAdminList;
1827         //public System.Collections.Specialized.StringDictionary DomainList;
1828         //public System.Collections.Specialized.StringDictionary DomainInProgressList;
1829         /// <summary>
1830         /// This instantiates a Active Directory scanning object
1831         /// </summary>
1832         ///
1833         public ScannerActiveDirectory()
1834         {
1835             DomainAdminList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
1836             DomainList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
1837             DomainInProgressList = new System.Collections.Concurrent.ConcurrentDictionary<String, String>();
1838         }
1839         /// <summary>
1840         /// This builds a server list from Active Directory
1841         /// </summary>
1842         ///
1843         public void ScanActiveDirectory()
1844         {
1845             int[] MaxThreads = { 0, 0 };
1846             int[] AvailableThreads = { 0, 0 };
1847             DomainList.TryAdd(Environment.UserDomainName, "0");
1848             Int32 OldDomainCount = 0;
1849             for (Int16 i = 0; i < 10; i++)
1850             {
1851                 Boolean Quit = true;
1852                 foreach (String Domain in DomainList.Keys)
1853                 {
1854                     if (!DomainList[Domain].Equals("Done", StringComparison.CurrentCultureIgnoreCase))
1855                         Quit = false;
1856                 }
1857                 if (Quit && i >= 2) break;
1858                 for (Int16 j = 0; j < 10; j++)
1859                 {
1860                     OldDomainCount = DomainList.Count;
1861                     foreach (String Domain in DomainList.Keys)
1862                     {
1863                         System.Threading.ThreadPool.QueueUserWorkItem(
1864                             new System.Threading.WaitCallback(ScanTrusts), Domain);
1865                     } //foreach domain get trusts
1866                     System.Threading.ThreadPool.GetMaxThreads(
1867                         out MaxThreads[0], out MaxThreads[1]);
1868                     System.Threading.ThreadPool.GetAvailableThreads(
1869                         out AvailableThreads[0], out AvailableThreads[1]);
1870                     for (Int16 k = 0;

```

```

1871         k < 10 && MaxThreads[0] - AvailableThreads[0] > 2 && DomainInProgressList.Count > 0;
1872         k++);
1873     {
1874         Tscan.Scan.UpdateProgress("10 Minute. " + k + "/10");
1875         System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
1876         System.Threading.ThreadPool.GetMaxThreads(
1877             out MaxThreads[0], out MaxThreads[1]);
1878         System.Threading.ThreadPool.GetAvailableThreads(
1879             out AvailableThreads[0], out AvailableThreads[1]);
1880     }
1881     if (OldDomainCount == DomainList.Count) break;
1882 }//for up to 10 trust scans
1883 foreach (String Domain in DomainList.Keys)
1884 {
1885     if (!DomainInProgressList[Domain].Equals(
1886         "InProgress", StringComparison.CurrentCultureIgnoreCase) &&
1887         !DomainList[Domain].Equals(
1888             "Done", StringComparison.CurrentCultureIgnoreCase) &&
1889         !DomainList[Domain].Equals(
1890             "10", StringComparison.CurrentCultureIgnoreCase))
1891     {
1892         Tscan.Scan.UpdateProgress(Domain);
1893         System.Threading.ThreadPool.QueueUserWorkItem(
1894             new System.Threading.WaitCallback(ScanDomain), Domain);
1895     }
1896 }
1897 System.Threading.ThreadPool.GetMaxThreads(
1898     out MaxThreads[0], out MaxThreads[1]);
1899 System.Threading.ThreadPool.GetAvailableThreads(
1900     out AvailableThreads[0], out AvailableThreads[1]);
1901 for (Int16 j = 0;
1902     j < 60 && MaxThreads[0] - AvailableThreads[0] > 2 && DomainInProgressList.Count > 0;
1903     j++)
1904 {
1905     Tscan.Scan.UpdateProgress("1 Hour Wait. " + j + "/60");
1906     System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
1907     System.Threading.ThreadPool.GetMaxThreads(
1908         out MaxThreads[0], out MaxThreads[1]);
1909     System.Threading.ThreadPool.GetAvailableThreads(
1910         out AvailableThreads[0], out AvailableThreads[1]);
1911 }
1912 }//for up to 10 loops of both trusts and undone domains
1913 //AD only stub
1914 System.IO.File.WriteAllText("Domainlist.txt", "");
1915 foreach (String Key in DomainList.Keys)
1916     System.IO.File.AppendAllText("Domainlist.txt", Key + Environment.NewLine);
1917 }
1918 /// <summary>ScanWinsAndSql
1919 /// This builds a list of servers from wins
1920 /// </summary>
1921 ///
1922 public void ScanWinsAndSql()
1923 {
1924     String StringTable = "";
1925     String StringHeader = "";
1926     String StringRow = "";
1927     Boolean HeaderDone = false;
1928     System.DirectoryServices.DirectoryEntry Root = new System.DirectoryServices.DirectoryEntry("WinNT:");
1929     foreach (System.DirectoryServices.DirectoryEntry Workgroup in Root.Children)
1930     {
1931         String WorkgroupName = Workgroup.Name;
1932         foreach (System.DirectoryServices.DirectoryEntry Computer in Workgroup.Children)
1933         {
1934             String ComputerName = Computer.Name;
1935             if (!ComputerName.Equals("Schema", StringComparison.CurrentCultureIgnoreCase))
1936             {
1937                 StringRow = "\"" + ComputerName + "\",\"" + WorkgroupName + "\",\"";
1938                 StringHeader = "Computer,\"Workgroup\",\"";
1939                 try
1940                 {
1941                     foreach (String Name in Computer.Properties.PropertyNames)
1942                     {
1943                         try
1944                         {
1945                             StringRow += "\"" + Computer.Properties[Name].Value + "\",\"";
1946                         }
1947                         catch
1948                         {
1949                             StringRow += "\"\\\",\"";
1950                         }
1951                         StringHeader += "\"" + Name + "\",\"";
1952                     }
1953                 }

```



```

1954         catch { } //sometimes there are no properties I think
1955         if (!HeaderDone) StringTable = StringHeader + Environment.NewLine;
1956         HeaderDone = true;
1957         StringTable += StringRow + Environment.NewLine;
1958         Tscan.Scan.ServerList.TryAdd(ComputerName, "");
1959     }
1960 }
1961 }
1962 Tscan.Scan.WriteToDisk("Wins.csv", StringTable);
1963 StringTable = "Computer,\"Instance\", \"Clustered\", \"Version\", \" + Environment.NewLine;
1964 System.Data.DataTable Table = System.Data.Sql.SqlDataSourceEnumerator.Instance.GetDataSources();
1965 foreach (System.Data.DataRow Row in Table.Rows)
1966 {
1967     StringTable += "\"\" + Row.ItemArray[0].ToString() + "\",\" +
1968         "\"\" + Row.ItemArray[1].ToString() + "\",\" +
1969         "\"\" + Row.ItemArray[2].ToString() + "\",\" +
1970         "\"\" + Row.ItemArray[3].ToString() + "\",\" + Environment.NewLine;
1971     //TryAdd
1972     if (!String.IsNullOrEmpty(Row.ItemArray[0].ToString()) &&
1973         !Tscan.Scan.ServerList.ContainsKey(Row.ItemArray[0].ToString()))
1974         Tscan.Scan.ServerList.TryAdd(Row.ItemArray[0].ToString(), "");
1975 }
1976 Tscan.Scan.WriteToDisk("SQL.csv", StringTable);
1977 }
1978 /// <summary>
1979 /// This builds a list of admins with bad passwords
1980 /// </summary>
1981 ///
1982 public void ScanTrusts(Object ObjectDomain)
1983 {
1984     String Domain = ObjectDomain.ToString();
1985     DomainInProgressList.TryAdd(Domain, "InProgress");
1986     try
1987     {
1988         Tscan.Scan.UpdateProgress(Domain);
1989         System.DirectoryServices.ActiveDirectory.Domain DomainDomain =
1990             System.DirectoryServices.ActiveDirectory.Domain.GetDomain(
1991                 new System.DirectoryServices.ActiveDirectory.DirectoryContext(
1992                     System.DirectoryServices.ActiveDirectory.DirectoryContextType.Domain, Domain));
1993         foreach (System.DirectoryServices.ActiveDirectory.Domain Child in DomainDomain.Children)
1994         {
1995             if (!DomainList.ContainsKey(Child.Name)) DomainList.TryAdd(Child.Name, "0");
1996         }
1997         foreach (System.DirectoryServices.ActiveDirectory.TrustRelationshipInformation Trust in
1998             DomainDomain.GetAllTrustRelationships())
1999         {
2000             if (!DomainList.ContainsKey(Trust.SourceName)) DomainList.TryAdd(Trust.SourceName, "0");
2001             if (!DomainList.ContainsKey(Trust.TargetName)) DomainList.TryAdd(Trust.TargetName, "0");
2002         }
2003         if (!DomainList.ContainsKey(DomainDomain.Parent.Name))
2004             DomainList.TryAdd(DomainDomain.Parent.Name, "0");
2005         //System.Threading.Thread.Sleep(TimeSpan.FromMinutes(1));
2006     }
2007     catch { }
2008     String Skip = "";
2009     DomainInProgressList.TryRemove(Domain, out Skip);
2010 }
2011 /// <summary>
2012 /// This builds a list of admins with bad passwords
2013 /// </summary>
2014 ///
2015 public void ScanAdmins()
2016 {
2017     Boolean Self = false;
2018     System.DirectoryServices.ActiveDirectory.Domain UserDomain =
2019         System.DirectoryServices.ActiveDirectory.Domain.GetCurrentDomain();
2020     while (UserDomain.Parent != null) UserDomain = UserDomain.Parent;
2021     String DomainDN = UserDomain.GetDirectoryEntry().Properties["distinguishedName"].ToString()
2022         .Replace("OU=Domain Controllers,", "");
2023     String StringFilter = "(&(objectClass=User))(memberOf=CN=Domain Admins,CN=Users,\"+DomainDN+)\");";
2024     String[] StringProperties = {"Name"};
2025     System.DirectoryServices.ActiveDirectory.Domain DomainDomain = UserDomain;
2026     System.DirectoryServices.DirectorySearcher Finder =
2027         new System.DirectoryServices.DirectorySearcher(
2028             DomainDomain.GetDirectoryEntry(), StringFilter, StringProperties);
2029     Finder.ClientTimeout = TimeSpan.FromMinutes(1);
2030     Finder.Asynchronous = true;
2031     Finder.PageSize = 1000;
2032     Finder.ServerPageTimeLimit = TimeSpan.FromMinutes(1);
2033     Finder.ServerTimeLimit = TimeSpan.FromMinutes(1);
2034     foreach (System.DirectoryServices.SearchResult Row in Finder.FindAll())
2035     {
2036         String Name = "Name";

```

```

2037         if (String.IsNullOrEmpty(Row.Properties[Name].ToString()))
2038         {
2039             foreach (String Pass in Tscan.Scan.WMIPasswords)
2040             {
2041                 if (Tscan.Scan.TestPassword(
2042                     UserDomain.FindDomainController().Name,
2043                     UserDomain.Name,
2044                     Row.Properties[Name].ToString(),
2045                     Pass).Equals(1))
2046                 {
2047                     DomainAdminList.TryAdd(
2048                         UserDomain.Name + "\\\" + Row.Properties[Name].ToString(),
2049                         Pass);
2050                     if (UserDomain.Name.Equals(
2051                         Environment.UserDomainName, StringComparison.CurrentCultureIgnoreCase) &&
2052                         Row.Properties[Name].ToString().Equals(
2053                             Environment.UserName, StringComparison.CurrentCultureIgnoreCase))
2054                         Self = true;
2055                 }
2056             }
2057         }
2058     }
2059     if (!Self)
2060     {
2061         String Current = DomainAdminList.Keys.GetEnumerator().Current.ToString();
2062         String Dom = Current.Split("\\\".ToCharArray())[0];
2063         String User = Current.Split("\\\".ToCharArray())[1];
2064         String Pass = DomainAdminList[Current];
2065         Tscan.Scan.RemoteExec.CompileService("\\\" + Dom + "\\\"", User, Pass);
2066     }
2067 }
2068
2069 /// <summary>
2070 /// This scans a single domain
2071 /// </summary>
2072 ///
2073 public void ScanDomain(Object ObjectDomain)
2074 {
2075     String Domain = ObjectDomain.ToString();
2076     DomainInProgressList.TryAdd(Domain, "InProgress");
2077     ScanDomain(Domain, "Computer");
2078     ScanDomain(Domain, "User");
2079     String Skip = "";
2080     DomainInProgressList.TryRemove(Domain, out Skip);
2081 }
2082
2083 /// <summary>
2084 /// This scans a single domain and a single type of object
2085 /// </summary>
2086 ///
2087 public void ScanDomain(String Domain, String Object)
2088 {
2089     //String StringDomain = Domain.ToString();
2090     String Names = "";
2091     String Values = "";
2092     String Table = "";
2093     Boolean HeaderDone = false;
2094     Boolean Fail = false;
2095     String StringFilter = "";
2096     String[] StringProperties;
2097     String[] StringPropertiesComputer = {"name", "StreetAddress", "PhysicalDeliveryOfficeName", "l", "st",
2098         "PostalCode", "co", "TelephoneNumber", "mail",
2099         "DNSHostName", "MacAddress", "OperatingSystem", "Description"};
2100     String[] StringPropertiesUser = {"name", "StreetAddress", "PhysicalDeliveryOfficeName", "l", "st",
2101         "PostalCode", "co", "TelephoneNumber", "mail",
2102         "GivenName", "sn", "EmployeeID", "LastLogon", "LastLogonTimestamp"};
2103     if (Object.Equals("Computer", StringComparison.CurrentCultureIgnoreCase))
2104     {
2105         StringFilter = "(objectClass=Computer)";
2106         StringProperties = StringPropertiesComputer;
2107         //File size for 200k users is 20MB
2108     }
2109     else if (Object.Equals("User", StringComparison.CurrentCultureIgnoreCase))
2110     {
2111         StringFilter = "(objectClass=User)";
2112         StringProperties = StringPropertiesUser;
2113         //mostrecentuser may be goose chase
2114     }
2115     else
2116     {
2117         IncrementDomainTries(Domain);
2118         return;
2119     }
2120 }
2121 try

```

```

2120     {
2121         //System.ArgumentException on getdomain
2122         System.DirectoryServices.ActiveDirectory.Domain DomainDomain =
2123             System.DirectoryServices.ActiveDirectory.Domain.GetDomain(
2124                 new System.DirectoryServices.ActiveDirectory.DirectoryContext(
2125                     System.DirectoryServices.ActiveDirectory.DirectoryContextType.Domain, Domain));
2126         System.DirectoryServices.DirectorySearcher Finder =
2127             new System.DirectoryServices.DirectorySearcher(
2128                 DomainDomain.GetDirectoryEntry(), StringFilter, StringProperties);
2129         Finder.ClientTimeout = TimeSpan.FromMinutes(1);
2130         Finder.Asynchronous = true;
2131         Finder.PageSize = 1000;
2132         Finder.ServerPageTimeLimit = TimeSpan.FromMinutes(1);
2133         Finder.ServerTimeLimit = TimeSpan.FromMinutes(1);
2134         foreach (System.DirectoryServices.SearchResult Row in Finder.FindAll())
2135         {
2136             foreach (String Name in Row.Properties.PropertyNames)
2137             {
2138                 Names += "\"" + Name + "\", ";
2139                 if (Row.Properties[Name] != null)
2140                 {
2141                     Values += "\"" + Tscan.Scan.ScrubString(Row.Properties[Name].ToString()) + "\", ";
2142                 }
2143                 else
2144                 {
2145                     Values += "\"\\\", ";
2146                 }
2147
2148                 //String foo = Row.Properties[Name].ToString();
2149             }
2150             if (Object.Equals("Computer", StringComparison.CurrentCultureIgnoreCase))
2151             {
2152                 if (String.IsNullOrEmpty(Row.Properties["DNSHostName"].ToString()))
2153                     Tscan.Scan.ServerList.TryAdd(Row.Properties["Name"].ToString(), "");
2154                 else Tscan.Scan.ServerList.TryAdd(Row.Properties["DNSHostName"].ToString(), "");
2155             }
2156         }
2157     }
2158     catch
2159     {
2160         IncrementDomainTries(Domain);
2161         Fail = true;
2162     }
2163     if (!HeaderDone) Table += Names + Environment.NewLine;
2164     Table += Values + Environment.NewLine;
2165     HeaderDone = true;
2166     Tscan.Scan.WriteToDisk(Domain + "_" + Object + ".csv", Table);
2167     if (String.IsNullOrEmpty(Table)) IncrementDomainTries(Domain);
2168     if (!String.IsNullOrEmpty(Table) && !Fail) DomainList[Domain] = "Done";
2169 }
2170 /// <summary>
2171 /// This helps track the number of times a domain has thrown an error
2172 /// </summary>
2173 ///
2174 public void IncrementDomainTries(String Domain)
2175 {
2176     Int16 Tries = 0;
2177     if (Int16.TryParse(DomainList[Domain], out Tries))
2178     {
2179         if (Tries < 10)
2180         {
2181             Tries++;
2182             DomainList[Domain] = Tries.ToString();
2183         }
2184     }
2185 }
2186 }
2187 }

```