

f5a8047587

...

BearFTP / BearFTP / Program.cs / <> Jump to

kolya5544 Final build. Still requires a bit of testing, but overall, it's reliable

History

0 contributors

1097 lines (997 sloc) 48.1 KB

...

```
1 using System;
2 using System.Net.Sockets;
3 using System.Threading;
4 using System.Text.RegularExpressions;
5 using System.Net;
6 using System.Collections.Generic;
7 using System.Security.Cryptography;
8 using System.Text;
9 using System.Diagnostics;
10 using System.IO;
11 using System.Net.Http;
12 using System.Net.Http.Headers;
13 using System.Linq;
14
15 namespace BearFTP
16 {
17     class Program
18     {
19         //CONFIG
20         public static int PortDef = 21;
21         public static int PortPasv = 1222;
22         public static string Hostname = "127.0.0.1";
23         public static string Token = "";
24         public static string Banner = "Welcome to FTP!";
25
26         public static bool Report = true;
27         public static bool Ban = true;
28         public static bool PunishScans = true;
29         public static bool AllowAnonymous = false;
30         public static bool PerIPLogs = false;
31
32         public static int Max_PerSecond = 5;
33         public static int Max_Total = 6;
34         public static int BanLength = 3600;
35         public static int MaxErrors = 6;
36         public static int BufferSize = 8192;
37
38         //IP TempBan list (hostname:seconds)
39         public static List<Ban> bans = new List<Ban>();
40
41         //An instance of config to extract values
42         public static Config config;
43
44         //Used because everybody likes random numbers.
45         public static Random rnd = new Random();
46         public static readonly HttpClient client = new HttpClient();
47         //List of all connected (to main port) clients
48         public static List<Client> connected = new List<Client>();
49
50         //Default directory. TODO: Implement directories
51         public static Directory root = new Directory();
52
53         //Current version
54         public static string _VERSION = "v0.3.0 BETA";
55
56         //Default log.
57         public static StreamWriter logfile = new StreamWriter("log.txt", true);
58
59         //Per-IP logs
60         public static List<StreamWriter> perips = new List<StreamWriter>();
61
62         //Dictionary of passvie clients (clients with PASV mode. Used to communicate directly later.)
63         public static Dictionary<Client, Connectivity> passives = new Dictionary<Client, Connectivity>();
64
65         //List of connections per second from hostname
66         public static List<Active> per_second = new List<Active>();
67         //List of overall connections from hostname
68         public static List<Active> actives = new List<Active>();
69
70         //List of overall connections to PASV
71         public static List<Active> pasv_actives = new List<Active>();
72
73         /// <summary>
74         /// Reports an IP
75         /// </summary>
76         /// <param name="hostname">IP to report</param>
77         /// <param name="comment">Logs or comments regarding report</param>
78         /// <param name="hacking">Is accused in hacking?</param>
```

```

79     /// <param name="brute">Is accused in brutng?</param>
80     /// <param name="webapp_h">Is accused in webapp hacking?</param>
81     /// <param name="scanning">Is accused in portscanning?</param>
82     /// <param name="ddos">Is accused in DDoS?</param>
83     /// <returns>A task to execute</returns>
84     public static async System.Threading.Tasks.Task ReportAsync(string hostname, string comment, bool hacking, bool brute, bool webapp_h, bool scanning, bool ddos)
85     {
86
87         string bad = "";
88         if (hacking)
89         {
90             bad += "15,";
91         }
92         if (brute)
93         {
94             bad += "18,5,";
95         }
96         if (webapp_h)
97         {
98             bad += "21,";
99         }
100        if (scanning)
101        {
102            bad += "14,";
103        }
104        if (ddos)
105        {
106            bad += "4,";
107        }
108        bad = bad.Substring(0, bad.Length - 1);
109        if (Report) {
110            try
111            {
112                using (var httpClient = new HttpClient())
113                {
114                    using (var request = new HttpRequestMessage(new HttpMethod("POST"), "https://api.abuseipdb.com/api/v2/report"))
115                    {
116                        request.Headers.TryAddWithoutValidation("Key", Token);
117                        request.Headers.TryAddWithoutValidation("Accept", "application/json");
118
119                        var contentList = new List<string>();
120                        contentList.Add($"ip={Uri.EscapeDataString(hostname)}");
121                        contentList.Add("categories=" + bad);
122                        contentList.Add($"comment={Uri.EscapeDataString(comment)}");
123                        request.Content = new StringContent(string.Join("&", contentList));
124                        request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/x-www-form-urlencoded");
125                        Console.WriteLine("=== REPORTING IP... " + hostname);
126                        var response = await httpClient.SendAsync(request);
127                        if (response.StatusCode == HttpStatusCode.OK)
128                        {
129                            Console.WriteLine("=== REPORTED IP " + hostname);
130                        }
131                        else
132                        {
133                            Console.WriteLine("=== ERROR WHILE REPORTING: " + response.StatusCode.ToString());
134                            Console.WriteLine("=== " + response.Content.ToString());
135                        }
136                    }
137                }
138            }
139            catch (Exception e)
140            {
141                Console.WriteLine(e.StackTrace);
142            }
143        }
144    }
145
146    /// <summary>
147    /// Hashes a string using md5
148    /// </summary>
149    /// <param name="input">String to hash</param>
150    /// <returns>Hashed string</returns>
151    public static string md5(string input)
152    {
153        MD5 md5 = MD5.Create();
154        byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);
155        byte[] hash = md5.ComputeHash(inputBytes);
156        StringBuilder sb = new StringBuilder();
157        for (int i = 0; i < hash.Length; i++)
158        {
159            sb.Append(hash[i].ToString("x2"));
160        }
161        return sb.ToString();
162    }
163
164    public static List<File> files = new List<File>();
165
166    /// <summary>
167    /// Writes to the StreamWriter as well as logging actions
168    /// </summary>
169    /// <param name="text">String to send over socket</param>
170    /// <param name="sw">StreamWriter of a TcpClient</param>
171    /// <param name="IP">Hostname of receiver</param>
172    /// <param name="perip">PerIP StreamWriter</param>
173    public static void LogWrite(string text, StreamWriter sw, string IP, StreamWriter perip = null)
174    {
175        Log(text.Trim().Replace("\r", String.Empty).Replace("\n", String.Empty).Trim(), "out", true, IP, perip);
176        sw.Write(text);
177    }

```

```

177
178 /// <summary>
179 /// Used to calculate and format the string for PASV mode
180 /// </summary>
181 /// <param name="port">Port of PASV</param>
182 /// <param name="host">Hostname (IP ONLY!)</param>
183 /// <returns>Formatted string</returns>
184 public static string PasvInit(int port, string host)
185 {
186     string actual_host = host.Replace('.', ',');
187     string actual_port = "";
188     int p1 = 0;
189     int p2 = 0;
190     for (int i = 0; i < 255; i++)
191     {
192         if (port - (256 * i) < 256)
193         {
194             p1 = i;
195             break;
196         }
197     }
198     p2 = port - (256 * p1);
199
200     actual_port = p1 + "," + p2;
201
202     return "(" + actual_host + "," + actual_port + ")";
203 }
204
205 /// <summary>
206 /// Checks using pastebin if a new version is out. Replace with your own URL
207 /// </summary>
208 /// <returns></returns>
209 public static bool CheckVersion()
210 {
211     using (var client = new WebClient())
212     {
213         try
214         {
215             var responseString = client.DownloadString("https://pastebin.com/raw/9dCZvME9");
216             if (responseString.Trim() != _VERSION)
217             {
218                 return false;
219             }
220             return true;
221         } catch { return false; }
222     }
223 }
224
225
226 /// <summary>
227 /// Logs text and prints it to console
228 /// </summary>
229 /// <param name="text">Text of a message</param>
230 /// <param name="dir">Either "in" for << or "out" for >></param>
231 /// <param name="date">Include date in format [MM/dd/yyyy HH:mm:ss] or not</param>
232 /// <param name="IP">IP Address to include before date (you can't have this true and date set to false)</param>
233 /// <param name="sw">PerIP StreamWriter handler</param>
234 public static void Log(string text, string dir, bool date = true, string IP = null, StreamWriter sw = null)
235 {
236     string Builder = "";
237     if (date)
238     {
239         if (IP == null)
240         {
241             Builder += "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + " ] ";
242         } else
243         {
244             Builder += "[" + IP + " " + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + " ] ";
245         }
246     }
247     if (dir == "in")
248     {
249         Builder += "<< ";
250     } else
251     {
252         Builder += ">> ";
253     }
254
255     Builder += text;
256
257     Builder = Regex.Replace(Builder, @"[\u0020-\u007E]", " ");
258
259     if (sw != null && PerIPLogs)
260     {
261         try
262         {
263             sw.WriteLine(Builder);
264         } catch
265         {
266
267         }
268     }
269
270     logfile.WriteLine(Builder);
271     Console.WriteLine(Builder);
272 }
273
274 static void Main(string[] args)

```

```

275 {
276     Console.OutputEncoding = Encoding.Unicode;
277     logfile.AutoFlush = true;
278     Log("Initialized server! >>", "in");
279     config = new Config("config.json");
280     PortDef = config.PortDef;
281     PortPasv = config.PortPasv;
282     Hostname = config.Hostname;
283     Token = config.Token;
284     Banner = config.Banner;
285     Report = config.Report;
286     Ban = config.Ban;
287     PunishScans = config.PunishScans;
288     AllowAnonymous = config.AllowAnonymous;
289     Max_PerSecond = config.Max_PerSecond;
290     Max_Total = config.Max_Total;
291     BanLength = config.BanLength;
292     MaxErrors = config.MaxErrors;
293     BufferSize = config.BufferSize;
294     PerIPLogs = config.PerIPLogs;
295
296     if (PortDef == PortPasv)
297     {
298         Console.ForegroundColor = ConsoleColor.Red;
299         Console.WriteLine("-> You are probably running a default/incorrect config! Please, edit it before starting the server");
300         Console.ResetColor();
301         Environment.Exit(1);
302     }
303
304
305
306     //Yes, it starts..
307     Console.WriteLine("- BearFTP OpenSource HoneyPot Server " + _VERSION + " -");
308     Console.WriteLine("- By IKTeam -> https://github.com/kolya5544/BearFTP -");
309     Console.WriteLine("Checking for updates...");
310     if (!CheckVersion())
311     {
312         Console.ForegroundColor = ConsoleColor.Red;
313         Console.WriteLine("----> You are *probably* running an outdated version of our software!");
314         Console.ResetColor();
315     } else
316     {
317         Console.WriteLine("You are running the latest version!");
318     }
319     Console.WriteLine("Running on " + Hostname + ":" + PortDef.ToString());
320     Console.WriteLine("PASV params: " + PasvInit(PortPasv, Hostname));
321     root.path = "/";
322     InitializeFiles();
323     TcpListener ftp = new TcpListener(PortDef);
324     TcpListener pasv = new TcpListener(PortPasv);
325     //Ban expiration handling.
326     new Thread(new ThreadStart(() => {
327         Thread.CurrentThread.IsBackground = true;
328
329         while (true)
330         {
331             Thread.Sleep(2000);
332             for (int i = 0; i < bans.Count; i++) {
333                 bans[i].time -= 2;
334                 if (bans[i].time <= 0)
335                 {
336                     bans.Remove(bans[i]);
337                     i--;
338                 }
339             }
340         }
341     })).Start();
342     //Connections per seconds (antibot) handling
343     new Thread(new ThreadStart(() => {
344         Thread.CurrentThread.IsBackground = true;
345
346         while (true)
347         {
348             Thread.Sleep(1000);
349             for (int i = 0; i < per_second.Count; i++)
350             {
351                 if (per_second[i].connected > 0)
352                 {
353                     per_second[i].connected -= 1;
354                 }
355             }
356             // Console.WriteLine("[DBG] Iterated per_second!");
357         }
358     })).Start();
359     ftp.Start();
360     pasv.Start();
361     new Thread(() =>
362     {
363         Thread.CurrentThread.IsBackground = true;
364         while (true)
365         {
366             TcpClient client = ftp.AcceptTcpClient();
367
368             NetworkStream ns = client.GetStream();
369             ns.ReadTimeout = 3000;
370             ns.WriteTimeout = 3000;
371             StreamReader sr = new StreamReader(ns);
372             StreamWriter sw = new StreamWriter(ns);

```

```

373
374
375 StreamWriter perip = null;
376
377 sw.AutoFlush = true;
378 string hostname = ((EndPoint)client.Client.RemoteEndPoint).Address.ToString();
379 if (System.IO.Directory.Exists("iplogs") && PerIPLogs)
380 {
381     try
382     {
383         if (!perips.Any(logs => ((FileStream)(logs.BaseStream)).Name.Contains(hostname)))
384         {
385             perip = new StreamWriter("iplogs/" + hostname + ".txt", true);
386             perip.AutoFlush = true;
387
388             perips.Add(perip);
389         } else
390         {
391             foreach (StreamWriter ip in perips)
392             {
393                 if (((FileStream)(ip.BaseStream)).Name.Contains(hostname))
394                 {
395                     perip = ip; break;
396                 }
397             }
398         }
399     } catch
400     {
401     }
402 }
403
404 if (Active.CheckExists(hostname, actives))
405 {
406     if (Active.GetConnections(hostname, actives) >= Max_Total)
407     {
408         client.Close();
409         if (Ban)
410         {
411             var aaa = new Ban();
412             aaa.hostname = hostname;
413             aaa.time = BanLength;
414             bans.Add(aaa);
415         }
416     } else
417     {
418         Active.SetConnections(hostname, actives, Active.GetConnections(hostname, actives) + 1);
419     }
420 }
421
422 else
423 {
424     actives.Add(new Active(hostname, 1));
425 }
426
427 if (Active.CheckExists(hostname, per_second))
428 {
429     if (Active.GetConnections(hostname, per_second) >= Max_PerSecond)
430     {
431         client.Close();
432         if (Ban)
433         {
434             var aaa = new Ban();
435             aaa.hostname = hostname;
436             aaa.time = BanLength;
437             bans.Add(aaa);
438         }
439     } else
440     {
441         Active.SetConnections(hostname, per_second, Active.GetConnections(hostname, per_second) + 1);
442     }
443 }
444
445 else
446 {
447     per_second.Add(new Active(hostname, 1));
448 }
449
450 try
451 {
452     if (bans.Any(ban => ban.hostname == hostname))
453     {
454         client.Close();
455     }
456 } catch
457 {
458 }
459
460
461 new Thread(new ThreadStart(() =>
462 {
463     Thread.CurrentThread.IsBackground = true;
464
465     bool triggered = false;
466     bool trigger2 = false;
467     Client c = new Client("null", "null", "null");
468     string username = "";
469     string password = "";

```

```

471     string directory = "/";
472     bool Authed = false;
473     bool passive = false;
474     int error = MaxErrors;
475
476
477     //AbuseDBIP.com API
478     bool hacking = false;
479     bool bruteforce = false;
480     bool webapp = false;
481     string comment = "";
482
483
484     bool banned = false;
485
486
487
488
489     try
490     {
491         Thread.Sleep(100);
492         Log("Connected - " + hostname, "in", true, hostname, perip);
493         LogWrite("220 " + Banner.Replace("%host%", Hostname) + "\r\n", sw, hostname, perip);
494
495         while (client.Connected)
496         {
497             Thread.Sleep(100);
498             //Receiving handler START
499             string ans = "";
500             bool flag = true;
501             bool upper = true;
502
503             while (flag)
504             {
505                 int a = sr.Read();
506                 if (upper)
507                 {
508                     ans += char.ToUpper((char)a);
509                 } else
510                 {
511                     ans += (char)a;
512                 }
513                 if (a == 13)
514                 {
515                     flag = false;
516                 }
517                 if (a == 0x20)
518                 {
519                     upper = false;
520                 }
521                 if (ans.Length > 128)
522                 {
523                     client.Close();
524                 }
525             }
526             ans = ans.Trim();
527             //Receiving handler END
528
529             //Command processing.
530             if (ans.Length >= 3) //We dont want dummies to spam/DDoS.
531             {
532                 Log(ans, "in", true, hostname, perip);
533             }
534             if (ans.StartsWith("CONNECT") || ans.StartsWith("GET http"))
535             {
536                 if (Ban)
537                 {
538                     var aaa = new Ban();
539                     aaa.hostname = hostname;
540                     aaa.time = Ban.Length;
541                     bans.Add(aaa);
542                     client.Close();
543                 }
544                 var a = ReportAsync(hostname, "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] " + "System scanning (Proxy judging) using CONNECT or GET");
545                 a.Start();
546
547
548             }
549             if (ans.Length > 128)
550             {
551                 client.Close();
552             }
553             if (ans.StartsWith("OPTS"))
554             {
555                 LogWrite("200 Encoding successfully changed!\r\n", sw, hostname, perip);
556             }
557             else if (ans.StartsWith("USER") && username.Length < 3 && !Authed)
558             {
559                 string temp = ans.Substring(5).Trim();
560                 Regex r = new Regex("^([a-zA-Z0-9]*)$");
561                 if (r.IsMatch(temp) && temp.Length < 32 && temp.Length > 1 && (temp != "anonymous" && !AllowAnonymous))
562                 {
563                     username = temp;
564                     LogWrite("331 This user is protected with password\r\n", sw, hostname, perip);
565                 }
566             }
567             else
568             {

```

```

569         LogWrite("530 Wrong username or/and password.\r\n", sw, hostname, perip);
570         if (temp.Length > 128)
571         {
572             client.Close();
573         }
574     }
575 }
576 else if (answ.StartsWith("PASS") && password.Length < 3 && !Authed)
577 {
578     string temp = answ.Substring(5).Trim();
579     if (temp.Length < 32 && temp.Length > 1)
580     {
581         password = temp;
582         if (password == "IEUser@" && PunishScans)
583         {
584             if (Ban)
585             {
586                 var aaa = new Ban();
587                 aaa.hostname = hostname;
588                 aaa.time = BanLength;
589                 bans.Add(aaa);
590                 client.Close();
591             }
592             var a = ReportAsync(hostname, "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] " + "System scanning (port scanning) using NMAP",
593                 a.Start());
594         }
595     }
596 }
597
598 LogWrite("230 Successful login.\r\n", sw, hostname, perip);
599 ns.ReadTimeout = 60000;
600 Authed = true;
601 c = new Client(username, password, hostname);
602
603 connected.Add(c);
604 }
605 else
606 {
607     LogWrite("530 Wrong username or/and password.\r\n", sw, hostname, perip);
608     if (temp.Length > 128)
609     {
610         client.Close();
611     }
612 }
613 }
614 else if (answ.Trim() == "SYST")
615 {
616     LogWrite("215 UNIX Type: L8\r\n", sw, hostname, perip);
617 }
618 else if (answ.Trim() == "FEAT")
619 {
620     LogWrite("502 Command unavailable.\r\n", sw, hostname, perip);
621 }
622 else if (answ.Trim() == "PWD")
623 {
624     LogWrite("257 \"" + directory + "\" is the current working directory\r\n", sw, hostname, perip);
625 }
626 else if (answ.Trim() == "PORT")
627 {
628     LogWrite("502 Command unavailable.\r\n", sw, hostname, perip);
629 }
630 else if (answ.Trim().StartsWith("TYPE"))
631 {
632     LogWrite("200 OK!\r\n", sw, hostname, perip);
633 }
634 else if (answ.Trim().StartsWith("STOR") && Authed)
635 {
636     Thread.Sleep(2000);
637     if (passives.ContainsKey(c))
638     {
639         Connectivity connn;
640         passives.TryGetValue(c, out connn);
641         if (connn.tcp.Connected)
642         {
643             Thread.Sleep(1000);
644             LogWrite("150 Ok to send data.\r\n", sw, hostname, perip);
645             Thread.Sleep(100);
646             List<byte> filess = new List<byte>();
647             var bytes = default(byte[]);
648             using (var memstream = new MemoryStream())
649             {
650                 var buffer = new byte[512];
651                 var bytesRead = default(int);
652                 while ((bytesRead = connn sr.BaseStream.Read(buffer, 0, buffer.Length)) > 0)
653                     memstream.Write(buffer, 0, bytesRead);
654                 bytes = memstream.ToArray();
655             }
656             System.IO.File.WriteAllBytes("dumps/dump_i" + rnd.Next(1, 2000000000).ToString() + ".txt", bytes);
657             Thread.Sleep(200);
658             LogWrite("226 Transfer complete!\r\n", sw, hostname, perip);
659         }
660     }
661     if (Ban)
662     {
663         var aaa = new Ban();
664         aaa.hostname = hostname;
665         aaa.time = BanLength;
666         bans.Add(aaa);
667         client.Close();
668     }
669 }

```

```

        }
        var a = ReportAsync(hostname, "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] " + "Unauthorized system access using FTP", true,
        a.Start());
    }
    else
    {
        client.Close();
        c.Connected = false;
    }
}
}
else if (answ.StartsWith("RETR") && Authed)
{
    Thread.Sleep(2000);
    string filename = answ.Substring(5).Trim().Replace("/", "");
    File aaaa = null;
    foreach (File aa in files)
    {
        if (aa.name == filename)
        {
            aaaa = aa;
        }
    }
    if (passives.ContainsKey(c) && aaaa != null)
    {
        Connectivity connn;
        passives.TryGetValue(c, out connn);
        if (connn.tcp.Connected)
        {
            Thread.Sleep(1000);
            LogWrite("150 Ok to send data.\r\n", sw, hostname, perip);
            Thread.Sleep(100);
            // byte[] file = aaaa.content;
            //Encoding.ASCII.GetChars(file);
            // connn.sw.Write(chars, 0, file.Length);
            // connn.tcp.Close();
            SendFile(aaaa, connn.sw);
            connn.tcp.Close();
            Thread.Sleep(200);
            LogWrite("226 Transfer complete!\r\n", sw, hostname, perip);

            if (Ban)
            {
                var aaa = new Ban();
                aaa.hostname = hostname;
                aaa.time = BanLength;
                bans.Add(aaa);
                client.Close();
            }
            var a = ReportAsync(hostname, "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] " + "Unauthorized system access using FTP", true,
            a.Start());
        }
    }
    else
    {
        client.Close();
        c.Connected = false;
    }
}
}
else if (answ.Trim() == "PASV" && Authed)
{
    if (Authed && !passive)
    {
        LogWrite("227 Entering Passive Mode " + PasvInit(PortPasv, Hostname) + "\r\n", sw, hostname, perip);
        c.passive = true;
    }
}
else if (answ.Trim().StartsWith("SIZE") && Authed)
{
    string filename = answ.Substring(5).Trim().Replace("/", "");
    File aaaa = null;
    foreach (File aa in files)
    {
        if (aa.name == filename)
        {
            aaaa = aa;
        }
    }
    if (aaaa != null)
    {
        LogWrite("213 " + aaaa.size.ToString(), sw, hostname, perip);
    }
}
else if (answ.Trim().StartsWith("LIST") && Authed)
{
    Thread.Sleep(1500);
    if (passives.ContainsKey(c))
    {
        Connectivity connn;
        passives.TryGetValue(c, out connn);
        if (connn.tcp.Connected)
        {
            string Builder = "";
            Builder += "drwxrwxrwx 5 root root 12288 Dec 1 16:51 .\r\n";

```



```

765         Builder += "drwxrwxrwx 5 root root 12288 Dec 1 16:51 ..\r\n";
766         int length = 5;
767         foreach (File file in files)
768         {
769             if (file.size.ToString().Length > length)
770             {
771                 length = file.size.ToString().Length;
772             }
773         }
774         foreach (File file in files)
775         {
776             Builder += file.chmod;
777             Builder += " " + rnd.Next(1, 9) + " ";
778             Builder += "root root ";
779             Builder += new string(' ', length - file.size.ToString().Length) + file.size.ToString();
780             Builder += " " + file.creation;
781             Builder += " " + file.name + "\r\n";
782         }
783         LogWrite("150 Here comes the directory listing.\r\n", sw, hostname, perip);
784         Thread.Sleep(100);
785         connn.sw.Write(Builder);
786         connn.tcp.Close();
787         Thread.Sleep(100);
788         LogWrite("226 Directory send OK\r\n", sw, hostname, perip);
789
790     }
791     else
792     {
793         client.Close();
794         c.Connected = false;
795     }
796 }
797
798 else if (answ.StartsWith("CMD"))
799 {
800     LogWrite("200 OK!\r\n", sw, hostname, perip);
801 }
802 else if (answ.StartsWith("CPFR"))
803 {
804     //Fun part: tricking random exploiters. Very "hackers"
805     triggered = true; //First level trigger
806     LogWrite("350 Need more information.\r\n", sw, hostname, perip);
807 }
808 else if (answ.Trim().StartsWith("CPTO") && triggered)
809 {
810     LogWrite("250 Need more information.\r\n", sw, hostname, perip);
811 }
812
813 else if (answ.Trim().StartsWith("AUTH"))
814 {
815     LogWrite("502 Please use plain FTP.\r\n", sw, hostname, perip); // We dont want them to use security.
816 }
817 else if (Authenticated && username == "admin" && md5(password) == "")
818 {
819     //Todo: admin cmds
820 }
821 else if (answ.Trim().StartsWith("CLNT"))
822 {
823     LogWrite("200 OK!\r\n", sw, hostname, perip);
824 }
825 else if (Authenticated && answ.Trim().StartsWith("NOOP"))
826 {
827     LogWrite("200 OK!\r\n", sw, hostname, perip);
828 } else if (Authenticated && answ.Trim().StartsWith("REST"))
829 {
830     LogWrite("502 There is no such command.\r\n", sw, hostname, perip);
831 }
832 else
833 {
834     if (answ.Length >= 3)
835     {
836         error--;
837         if (error <= 0)
838         {
839             client.Close();
840         }
841     }
842 }
843
844 if (answ.Contains("php") && triggered)
845 {
846     trigger2 = true; //Second level trigger
847 }
848 if (trigger2)
849 {
850     LogWrite("110 Illegal activity was detected.\r\n", sw, hostname, perip);
851     LogWrite("110 Please, log off now.\r\n", sw, hostname, perip);
852     if (Ban)
853     {
854         var aaa = new Ban();
855         aaa.hostname = hostname;
856         aaa.time = BanLength;
857         bans.Add(aaa);
858         client.Close();
859     }
860     var a = ReportAsync(hostname, "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] " + "RCE Attempt at 21 port using ProFTPD exploit", true,
861     a.Start();
862

```

```

863         }
864     }
865 }
866
867     }
868
869     }
870     catch (Exception e)
871     {
872         client.Close();
873         c.Connected = false;
874     }
875     Active.SetConnections(hostname, actives, Active.GetConnections(hostname, actives) - 1);
876     if (PerIPLogs)
877     {
878         try
879         {
880             perips.Remove(perip);
881             perip.Close();
882         } catch
883         {
884             }
885     }
886 }
887 }
888 }).Start();
889 }
890 }).Start();
891 new Thread(() =>
892 {
893
894     //THIS IS A TOTAL MESS. DON'T TOUCH IT UNLESS YOU REALLY WANT TO EDIT PASV MODE ANYHOW.
895     //Shortly how it works:
896     //1. Client connects to main port.
897     //2. Initiates PASV mode
898     //3. He is then set as "passive"
899     //4. He connects to THIS one.
900     //5. He is then assigned a Connectivity based of his hostname and either or not he is still connected.
901     //6. This basically creates a link between Main socket and Pasv socket, allowing Main to access Pasv using a Dictionary.
902     Thread.CurrentThread.IsBackground = true;
903
904     Client cll = new Client(null, null, null);
905     while (true)
906     {
907         TcpClient client = pasv.AcceptTcpClient();
908         NetworkStream ns = client.GetStream();
909         ns.ReadTimeout = 3000;
910         ns.WriteTimeout = 3000;
911         StreamReader sr = new StreamReader(ns);
912         StreamWriter sw = new StreamWriter(ns);
913
914         sw.AutoFlush = true;
915         string hostname = ((IPEndPoint)client.Client.RemoteEndPoint).Address.ToString();
916
917         try
918         {
919             if (bans.Any(ban => ban.hostname == hostname))
920             {
921                 client.Close();
922             }
923         }
924         catch
925         {
926         }
927     }
928
929     if (Active.CheckExists(hostname, pasv_actives))
930     {
931         if (Active.GetConnections(hostname, pasv_actives) >= Max_Total)
932         {
933             client.Close();
934             if (Ban)
935             {
936                 var aaa = new Ban();
937                 aaa.hostname = hostname;
938                 aaa.time = BanLength;
939                 bans.Add(aaa);
940             }
941         }
942         else
943         {
944             Active.SetConnections(hostname, pasv_actives, Active.GetConnections(hostname, pasv_actives) + 1);
945         }
946     }
947     else
948     {
949         pasv_actives.Add(new Active(hostname, 1));
950     }
951     Thread user = new Thread(new ThreadStart(() =>
952     {
953         Thread.CurrentThread.IsBackground = true;
954
955         Client c = new Client("1", "2", "3");
956
957
958         try
959         {
960

```

```

961         bool ispresent = false;
962         foreach (Client cl in connected)
963         {
964             if (cl.hostname == hostname && cl.Connected)
965             {
966                 c = cl;
967                 ispresent = true;
968             }
969         }
970         if (!ispresent)
971         {
972             client.Close();
973         }
974     }
975     else
976     {
977         Connectivity ca = new Connectivity();
978         ca.sr = sr;
979         ca.sw = sw;
980         ca.tcp = client;
981         passives.Add(c, ca);
982         /* while (client.Connected)
983         {
984             Thread.Sleep(3000);
985         }*/
986         for (int i = 0; client.Connected; i++)
987         {
988             Thread.Sleep(1000);
989             if (i >= 120)
990             {
991                 client.Close();
992                 passives.Remove(c);
993             }
994         }
995         client.Close();
996         passives.Remove(c);
997     }
998 }
999 catch (Exception e)
1000 {
1001     if (!e.Message.StartsWith("An item"))
1002     {
1003         client.Close();
1004         passives.Remove(c);
1005     }
1006 }
1007 Active.SetConnections(hostname, pasv_actives, Active.GetConnections(hostname, pasv_actives) - 1);
1008 }
1009
1010 ));
1011 user.Start();
1012
1013
1014     }
1015 }).Start();
1016 while (true)
1017 {
1018     string ok = Console.ReadLine();
1019     //TODO: Internal command handler
1020 }
1021
1022 }
1023 /// <summary>
1024 /// Initializes files for LIST or RETR.
1025 /// </summary>
1026 private static void InitializeFiles()
1027 {
1028     if (config.files.Count > 0)
1029     {
1030         foreach (CJSON_FILE json in config.files)
1031         {
1032             if (!json.Content.StartsWith("---"))
1033             {
1034                 File file = new File(json.Name, json.Content.Length, "-rw-r--r--", "Dec 1 15:11", json.Content, root);
1035                 files.Add(file);
1036             } else
1037             {
1038                 try
1039                 {
1040                     var filecontents = System.IO.File.ReadAllBytes(json.Content.Substring(3, json.Content.Length - 3));
1041                     File file = new File(json.Name, filecontents.Length, "-rw-r--r--", "Dec 1 15:11", filecontents, root);
1042                     files.Add(file);
1043                 } catch (Exception e)
1044                 {
1045                     Console.WriteLine(e.Message);
1046                 }
1047             }
1048         }
1049     }
1050 }
1051 } else
1052 {
1053     File file = new File("readme.txt", 3, "-rw-r--r--", "Dec 1 15:10", "Hi!", root);
1054     files.Add(file);
1055 }
1056
1057 }
1058

```

```

1059     /// <summary>
1060     /// Sends contents of files in 2 kilobyte packs
1061     /// </summary>
1062     /// <param name="file">File to send</param>
1063     /// <param name="sw">Actual StreamWriter of PASV mode</param>
1064     public static void SendFile(File file, StreamWriter sw)
1065     {
1066         if (file.size <= BufferSize)
1067         {
1068             sw.BaseStream.Write(file.content, 0, file.size);
1069         } else
1070         {
1071             //Ok boomer
1072             //1. We calculate amount of steps (a.k.a how much should we do the loop)
1073             //2. We calculate offtop based on steps we already passed
1074             //3. We take BUFFERSIZE bytes since that offtop and send them.....
1075             //it's hard but here's the actual code:
1076
1077             int Steps = 0;
1078             int Offtop = 0;
1079             int Leftover = 0;
1080
1081             byte[] buffer = new byte[BufferSize];
1082             Steps = Math.DivRem(file.size, BufferSize, out Leftover);
1083             for(Offtop = 0; Offtop<Steps; Offtop++)
1084             {
1085                 Array.Copy(file.content, Offtop * BufferSize, buffer, 0, BufferSize);
1086                 sw.BaseStream.Write(buffer, 0, buffer.Length);
1087                 Thread.Sleep(50); //Trying to limit possible attacks.
1088             }
1089             var last = new byte[file.size - Offtop * BufferSize];
1090             Array.Copy(file.content, file.size - Leftover, last, 0, Leftover);
1091             sw.BaseStream.Write(last, 0, last.Length);
1092             Thread.Sleep(50);
1093             return;
1094         }
1095     }
1096 }
1097 }

```

