```
using System;
using System.Net.Sockets;
using System.Threading;
using System.Text.RegularExpressions;
using System.Net;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Linq;

namespace BearFTP
{
    class Program
    {
        //CONFIG
        public static int PortDef = 21;
        public static int PortPasv = 1222;
        public static string Hostname = "127.0.0.1";
        public static string Token = "";
        public static string Banner = "Welcome to FTP!";

        public static bool Report = true;
        public static bool Ban = true;
        public static bool PunishScans = true;
        public static bool AllowAnonymous = false;
        public static bool PerIPLogs = false;

        public static int Max_PerSecond = 5;
        public static int Max_Total = 6;
        public static int BanLength = 3600;
        public static int MaxErrors = 6;
        public static int BufferSize = 8192;

        //IP TempBan list (hostname:seconds)
        public static List<Ban> bans = new List<Ban>();

        //An instance of config to extract values
        public static Config config;

        //Used because everybody likes random numbers.
        public static Random rnd = new Random();
        public static readonly HttpClient client = new HttpClient();
        //List of all connected (to main port) clients
        public static List<Client> connected = new List<Client>();

        //Default directory. TODO: Implement directories
        public static Directory root = new Directory();

        //Current version
        public static string _VERSION = "v0.3.0 BETA";

        //Default log.
        public static StreamWriter logfile = new StreamWriter("log.txt", true);

        //Per-IP logs
        public static List<StreamWriter> perips = new List<StreamWriter>();

        //Dictionary of passvie clients (clients with PASV mode. Used to communicate directly later.)
        public static Dictionary<Client, Connectivity> passives = new Dictionary<Client, Connectivity>();

        //List of connections per second from hostname
        public static List<Active> per_second = new List<Active>();
        //List of overall connections from hostname
        public static List<Active> actives = new List<Active>();

        //List of overall connections to PASV
        public static List<Active> pasv_actives = new List<Active>();

        /// <summary>
        /// Reports an IP
        /// </summary>
        /// <param name="hostname">IP to report</param>
        /// <param name="comment">Logs or comments regarding report</param>
        /// <param name="hacking">Is accused in hacking?</param>
```

```
        /// <param name="brute">Is accused in bruting?</param>
        /// <param name="webapp_h">Is accused in webapp hacking?</param>
        /// <param name="scanning">Is accused in portscanning?</param>
        /// <param name="ddos">Is accused in DDoS</param>
        /// <returns>A task to execute</returns>
        public static async System.Threading.Tasks.Task ReportAsync(string hostname, string comment, bool hacking, bool brute, bool webapp_h, bool scanning, bool ddos)
        {

            string bad = "";
            if (hacking)
            {
                bad += "15,";
            }
            if (brute)
            {
                bad += "18,5,";
            }
            if (webapp_h)
            {
                bad += "21,";
            }
            if (scanning)
            {
                bad += "14,";
            }
            if (ddos)
            {
                bad += "4,";
            }
            bad = bad.Substring(0, bad.Length - 1);
            if (Report) {
                try
                {
                    using (var httpClient = new HttpClient())
                    {
                        using (var request = new HttpRequestMessage(new HttpMethod("POST"), "https://api.abuseipdb.com/api/v2/report"))
                        {
                            request.Headers.TryAddWithoutValidation("Key", Token);
                            request.Headers.TryAddWithoutValidation("Accept", "application/json");

                            var contentList = new List<string>();
                            contentList.Add($"ip={Uri.EscapeDataString(hostname)}");
                            contentList.Add("categories=" + bad);
                            contentList.Add($"comment={Uri.EscapeDataString(comment)}");
                            request.Content = new StringContent(string.Join("&", contentList));
                            request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/x-www-form-urlencoded");
                            Console.WriteLine("=== REPORTING IP.... " + hostname);
                            var response = await httpClient.SendAsync(request);
                            if (response.StatusCode == HttpStatusCode.OK)
                            {
                                Console.WriteLine("=== REPORTED IP " + hostname);
                            }
                            else
                            {
                                Console.WriteLine("=== ERROR WHILE REPORTING: " + response.StatusCode.ToString());
                                Console.WriteLine("=== " + response.Content.ToString());
                            }
                        }
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.StackTrace);
                }
            }

        }
        /// <summary>
        /// Hashes a string using md5
        /// </summary>
        /// <param name="input">String to hash</param>
        /// <returns>Hashed string</returns>
        public static string md5(string input)
        {
            MD5 md5 = MD5.Create();
            byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);
            byte[] hash = md5.ComputeHash(inputBytes);
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < hash.Length; i++)
            {
                sb.Append(hash[i].ToString("x2"));
            }
            return sb.ToString();
        }
        public static List<File> files = new List<File>();

        /// <summary>
        /// Writes to the StreamWriter as well as logging actions
        /// </summary>
        /// <param name="text">String to send over socket</param>
        /// <param name="sw">StreamWriter of a TCPClient</param>
        /// <param name="IP">Hostname of receiver</param>
        /// <param name="perip">PerIP StreamWriter</param>
        public static void LogWrite(string text, StreamWriter sw, string IP, StreamWriter perip = null)
        {
            Log(text.Trim().Replace("\r", String.Empty).Replace("\n", String.Empty).Trim(), "out", true, IP, perip);
            sw.Write(text);
        }
```

```
        /// <summary>
        /// Used to calculate and format the string for PASV mode
        /// </summary>
        /// <param name="port">Port of PASV</param>
        /// <param name="host">Hostname (IP ONLY!)</param>
        /// <returns>Formatted string</returns>
        public static string PasvInit(int port, string host)
        {
            string actual_host = host.Replace('.', ',');
            string actual_port = "";
            int p1 = 0;
            int p2 = 0;
            for (int i = 0; i < 255; i++)
            {
                if (port - (256 * i) < 256)
                {
                    p1 = i;
                    break;
                }
            }
            p2 = port - (256 * p1);

            actual_port = p1 + "," + p2;

            return "(" + actual_host + "," + actual_port + ")";
        }

        /// <summary>
        /// Checks using pastebin if a new version is out. Replace with your own URL
        /// </summary>
        /// <returns></returns>
        public static bool CheckVersion()
        {
            using (var client = new WebClient())
            {
                try
                {
                    var responseString = client.DownloadString("https://pastebin.com/raw/9dCZvME9");
                    if (responseString.Trim() != _VERSION)
                    {
                        return false;
                    }
                    return true;
                } catch { return false; }

            }
        }

        /// <summary>
        /// Logs text and prints it to console
        /// </summary>
        /// <param name="text">Text of a message</param>
        /// <param name="dir">Either "in" for << or "out" for >></param>
        /// <param name="date">Include date in format [MM/dd/yyyy HH:mm:ss] or not</param>
        /// <param name="IP">IP Address to include before date (you can't have this true and date set to false)</param>
        /// <param name="sw">PerIP StreamWriter handler</param>
        public static void Log(string text, string dir, bool date = true, string IP = null, StreamWriter sw = null)
        {
            string Builder = "";
            if (date)
            {
                if (IP == null)
                {
                    Builder += "[" + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] ";
                } else
                {
                    Builder += "[" + IP + " " + DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss") + "] ";
                }
            }
            if (dir == "in")
            {
                Builder += "<< ";
            } else
            {
                Builder += ">> ";
            }

            Builder += text;

            Builder = Regex.Replace(Builder, @"[^\u0020-\u007E]", " ");

            if (sw != null && PerIPLogs)
            {
                try
                {
                    sw.WriteLine(Builder);
                } catch
                {

                }
            }

            logfile.WriteLine(Builder);
            Console.WriteLine(Builder);
        }

        static void Main(string[] args)
```

```
        {
            Console.OutputEncoding = Encoding.Unicode;
            logfile.AutoFlush = true;
            Log("Initialized server! >>", "in");
            config = new Config("config.json");
            PortDef = config.PortDef;
            PortPasv = config.PortPasv;
            Hostname = config.Hostname;
            Token = config.Token;
            Banner = config.Banner;
            Report = config.Report;
            Ban = config.Ban;
            PunishScans = config.PunishScans;
            AllowAnonymous = config.AllowAnonymous;
            Max_PerSecond = config.Max_PerSecond;
            Max_Total = config.Max_Total;
            BanLength = config.BanLength;
            MaxErrors = config.MaxErrors;
            BufferSize = config.BufferSize;
            PerIPLogs = config.PerIPLogs;

            if (PortDef == PortPasv)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("-> You are probably running a default/incorrect config! Please, edit it before starting the server");
                Console.ResetColor();
                Environment.Exit(1);
            }



            //Yes, it starts..
            Console.WriteLine("- BearFTP OpenSource HoneyPot Server " + _VERSION + " -");
            Console.WriteLine("- By IKTeam -> https://github.com/kolya5544/BearFTP -");
            Console.WriteLine("Checking for updates...");
            if (!CheckVersion())
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("----> You are *probably* running an outdated version of our software!");
                Console.ResetColor();
            } else
            {
                Console.WriteLine("You are running the latest version!");
            }
            Console.WriteLine("Running on " + Hostname + ":" + PortDef.ToString());
            Console.WriteLine("PASV params: " + PasvInit(PortPasv, Hostname));
            root.path = "/";
            InitializeFiles();
            TcpListener ftp = new TcpListener(PortDef);
            TcpListener pasv = new TcpListener(PortPasv);
            //Ban expiration handling.
            new Thread(new ThreadStart(() => {
                Thread.CurrentThread.IsBackground = true;

                while (true)
                {
                    Thread.Sleep(2000);
                    for (int i = 0; i<bans.Count; i++) {
                        bans[i].time -= 2;
                        if (bans[i].time <= 0)
                        {
                            bans.Remove(bans[i]);
                            i--;
                        }
                    }
                }
            })).Start();
            //Connections per seconds (antibot) handling
            new Thread(new ThreadStart(() => {
                Thread.CurrentThread.IsBackground = true;

                while (true)
                {
                    Thread.Sleep(1000);
                    for (int i = 0; i < per_second.Count; i++)
                    {
                        if (per_second[i].connected > 0)
                        {
                            per_second[i].connected -= 1;
                        }
                    }
                 //   Console.WriteLine("[DBG] Iterated per_second!");
                }
            })).Start();
            ftp.Start();
            pasv.Start();
            new Thread(() =>
            {
                Thread.CurrentThread.IsBackground = true;
                while (true)
                {
                    TcpClient client = ftp.AcceptTcpClient();

                    NetworkStream ns = client.GetStream();
                    ns.ReadTimeout = 3000;
                    ns.WriteTimeout = 3000;
                    StreamReader sr = new StreamReader(ns);
                    StreamWriter sw = new StreamWriter(ns);
```

```
                    StreamWriter perip = null;

                    sw.AutoFlush = true;
                    string hostname = ((IPEndPoint)client.Client.RemoteEndPoint).Address.ToString();
                    if (System.IO.Directory.Exists("iplogs") && PerIPLogs)
                    {
                        try
                        {
                            if (!perips.Any(logs => ((FileStream)(logs.BaseStream)).Name.Contains(hostname)))
                            {
                                perip = new StreamWriter("iplogs/" + hostname + ".txt", true);
                            perip.AutoFlush = true;

                                perips.Add(perip);
                            } else
                            {
                                foreach (StreamWriter ip in perips)
                                {
                                    if (((FileStream)(ip.BaseStream)).Name.Contains(hostname))
                                    {
                                        perip = ip; break;
                                    }
                                }
                            }
                        }
                        catch
                        {

                        }
                    }
                    if (Active.CheckExists(hostname, actives))
                    {
                        if (Active.GetConnections(hostname, actives) >= Max_Total)
                        {
                            client.Close();
                            if (Ban)
                            {
                                var aaa = new Ban();
                                aaa.hostname = hostname;
                                aaa.time = BanLength;
                                bans.Add(aaa);
                            }
                        }
                        else
                        {
                            Active.SetConnections(hostname, actives, Active.GetConnections(hostname, actives) + 1);
                        }
                    }
                    else
                    {
                        actives.Add(new Active(hostname, 1));
                    }

                    if (Active.CheckExists(hostname, per_second))
                    {
                        if (Active.GetConnections(hostname, per_second) >= Max_PerSecond)
                        {
                            client.Close();
                            if (Ban)
                            {
                                var aaa = new Ban();
                                aaa.hostname = hostname;
                                aaa.time = BanLength;
                                bans.Add(aaa);
                            }
                        }
                        else
                        {
                            Active.SetConnections(hostname, per_second, Active.GetConnections(hostname, per_second) + 1);
                        }
                    }
                    else
                    {
                        per_second.Add(new Active(hostname, 1));

                    }
                    try
                    {
                        if (bans.Any(ban => ban.hostname == hostname))
                        {
                            client.Close();
                        }
                    }
                    catch
                    {

                    }

                    new Thread(new ThreadStart(() =>
                    {
                        Thread.CurrentThread.IsBackground = true;

                        bool triggered = false;
                        bool trigger2 = false;
                        Client c = new Client("null", "null", "null");
                        string username = "";
                        string password = "";
```

```
                    string directory = "/";
                    bool Authed = false;
                    bool passive = false;
                    int error = MaxErrors;


                    //AbuseDBIP.com API
                    bool hacking = false;
                    bool bruteforce = false;
                    bool webapp = false;
                    string comment = "";


                    bool banned = false;




                    try
                    {
                        Thread.Sleep(100);
                        Log("Connected - " + hostname, "in", true, hostname, perip);
                        LogWrite("220 " + Banner.Replace("%host%", Hostname) + "\r\n", sw, hostname, perip);

                        while (client.Connected)
                        {
                            Thread.Sleep(100);
                            //Receiving handler START
                            string answ = "";
                            bool flag = true;
                            bool upper = true;

                            while (flag)
                            {
                                int a = sr.Read();
                                if (upper)
                                {
                                    answ += char.ToUpper((char)a);
                                } else
                                {
                                    answ += (char)a;
                                }
                                if (a == 13)
                                {
                                    flag = false;
                                }
                                if (a == 0x20)
                                {
                                    upper = false;
                                }
                                if (answ.Length > 128)
                                {
                                    client.Close();
                                }
                            }
                            answ = answ.Trim();
                            //Receiving handler END

                            //Command processing.
                            if (answ.Length >= 3) //We dont want dummies to spam/DDoS.
                            {
                                Log(answ, "in", true, hostname, perip);
                            }
                            if (answ.StartsWith("CONNECT") || answ.StartsWith("GET http"))
                            {
                                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") + "] " + "System scanning (Proxy judging) using CONNECT or GET
                                a.Start();


                            }
                            if (answ.Length > 128)
                            {
                                client.Close();
                            }
                            if (answ.StartsWith("OPTS"))
                            {
                                LogWrite("200 Encoding successfully changed!\r\n", sw, hostname, perip);
                            }
                            else if (answ.StartsWith("USER") && username.Length < 3 && !Authed)
                            {
                                string temp = answ.Substring(5).Trim();
                                Regex r = new Regex("^[a-zA-Z0-9]*$");
                                if (r.IsMatch(temp) && temp.Length < 32 && temp.Length > 1 && (temp != "anonymous" && !AllowAnonymous))
                                {
                                    username = temp;
                                    LogWrite("331 This user is protected with password\r\n", sw, hostname, perip);

                                }
                                else
                                {
```

```
                LogWrite("530 Wrong username or/and password.\r\n", sw, hostname, perip);
                if (temp.Length > 128)
                {
                    client.Close();
                }
            }
        }
        else if (answ.StartsWith("PASS") && password.Length < 3 && !Authed)
        {
            string temp = answ.Substring(5).Trim();
            if (temp.Length < 32 && temp.Length > 1)
            {
                password = temp;
                if (password == "IEUser@" && PunishScans)
                {
                    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") + "] " + "System scanning (port scanning) using NMAP",
                    a.Start();



                }
                LogWrite("230 Successful login.\r\n", sw, hostname, perip);
                ns.ReadTimeout = 60000;
                Authed = true;
                c = new Client(username, password, hostname);

                connected.Add(c);
            }
            else
            {
                LogWrite("530 Wrong username or/and password.\r\n", sw, hostname, perip);
                if (temp.Length > 128)
                {
                    client.Close();
                }
            }
        }
        else if (answ.Trim() == "SYST")
        {
            LogWrite("215 UNIX Type: L8\r\n", sw, hostname, perip);
        }
        else if (answ.Trim() == "FEAT")
        {
            LogWrite("502 Command unavailable.\r\n", sw, hostname, perip);
        }
        else if (answ.Trim() == "PWD")
        {
            LogWrite("257 \"" + directory + "\" is the current working directory\r\n", sw, hostname, perip);
        }
        else if (answ.Trim() == "PORT")
        {
            LogWrite("502 Command unavailable.\r\n", sw, hostname, perip);
        }
        else if (answ.Trim().StartsWith("TYPE"))
        {
            LogWrite("200 OK!\r\n", sw, hostname, perip);
        }
        else if (answ.Trim().StartsWith("STOR") && Authed)
        {
            Thread.Sleep(2000);
            if (passives.ContainsKey(c))
            {
                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);
                    List<byte> filess = new List<byte>();
                    var bytes = default(byte[]);
                    using (var memstream = new MemoryStream())
                    {
                        var buffer = new byte[512];
                        var bytesRead = default(int);
                        while ((bytesRead = connn.sr.BaseStream.Read(buffer, 0, buffer.Length)) > 0)
                            memstream.Write(buffer, 0, bytesRead);
                        bytes = memstream.ToArray();
                    }
                    System.IO.File.WriteAllBytes("dumps/dump_i" + rnd.Next(1, 2000000000).ToString() + ".txt", bytes);
                    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.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";
```

```
                    Builder += "drwxrwxrwx 5 root root 12288 Dec  1 16:51 ..\r\n";
                    int length = 5;
                    foreach (File file in files)
                    {
                        if (file.size.ToString().Length > length)
                        {
                            length = file.size.ToString().Length;
                        }
                    }
                    foreach (File file in files)
                    {
                        Builder += file.chmod;
                        Builder += " " + rnd.Next(1, 9) + " ";
                        Builder += "root root ";
                        Builder += new string(' ', length - file.size.ToString().Length) + file.size.ToString();
                        Builder += " " + file.creation;
                        Builder += " " + file.name + "\r\n";
                    }
                    LogWrite("150 Here comes the directory listing.\r\n", sw, hostname, perip);
                    Thread.Sleep(100);
                    connn.sw.Write(Builder);
                    connn.tcp.Close();
                    Thread.Sleep(100);
                    LogWrite("226 Directory send OK\r\n", sw, hostname, perip);

                }
                else
                {
                    client.Close();
                    c.Connected = false;
                }
            }
        }
        else if (answ.StartsWith("CWD"))
        {
            LogWrite("200 OK!\r\n", sw, hostname, perip);
        }
        else if (answ.StartsWith("CPFR"))
        {
            //Fun part: tricking random exploiters. Very "hackers"
            triggered = true; //First level trigger
            LogWrite("350 Need more information.\r\n", sw, hostname, perip);
        }
        else if (answ.Trim().StartsWith("CPTO") && triggered)
        {
            LogWrite("250 Need more information.\r\n", sw, hostname, perip);

        }
        else if (answ.Trim().StartsWith("AUTH"))
        {
            LogWrite("502 Please use plain FTP.\r\n", sw, hostname, perip); // We dont want them to use security.
        }
        else if (Authed && username == "admin" && md5(password) == "")
        {
            //Todo: admin cmds
        }
        else if (answ.Trim().StartsWith("CLNT"))
        {
            LogWrite("200 OK!\r\n", sw, hostname, perip);
        }
        else if (Authed && answ.Trim().StartsWith("NOOP"))
        {
            LogWrite("200 OK!\r\n", sw, hostname, perip);
        } else if (Authed && answ.Trim().StartsWith("REST"))
        {
            LogWrite("502 There is no such command.\r\n", sw, hostname, perip);
        }
        else
        {
            if (answ.Length >= 3)
            {
                error--;
                if (error <= 0)
                {
                    client.Close();
                }

            }
        }
        if (answ.Contains("php") && triggered)
        {
            trigger2 = true; //Second level trigger
        }
        if (trigger2)
        {
            LogWrite("110 Illegal activity was detected.\r\n", sw, hostname, perip);
            LogWrite("110 Please, log off now.\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") + "] " + "RCE Attempt at 21 port using ProFTPd exploit", true,
            a.Start();

```

```
                            }

                        }

                    }
                    catch (Exception e)
                    {
                        client.Close();
                        c.Connected = false;
                    }
                    Active.SetConnections(hostname, actives, Active.GetConnections(hostname, actives) - 1);
                    if (PerIPLogs)
                    {
                        try
                        {
                            perips.Remove(perip);
                            perip.Close();
                        } catch
                        {

                        }
                    }
                }
                )).Start();
            }
        }).Start();
        new Thread(() =>
        {

            //THIS IS A TOTAL MESS. DON'T TOUCH IT UNLESS YOU REALLY WANT TO EDIT PASV MODE ANYHOW.
            //Shortly how it works:
            //1. Client connects to main port.
            //2. Initiates PASV mode
            //3. He is then set as "passive"
            //4. He connects to THIS one.
            //5. He is then assigned a Connectivity based of his hostname and either or not he is still connected.
            //6. This basically creates a link between Main socket and Pasv socket, allowing Main to access Pasv using a Dictionary.
            Thread.CurrentThread.IsBackground = true;

            Client cll = new Client(null, null, null);
            while (true)
            {
                TcpClient client = pasv.AcceptTcpClient();
                NetworkStream ns = client.GetStream();
                ns.ReadTimeout = 3000;
                ns.WriteTimeout = 3000;
                StreamReader sr = new StreamReader(ns);
                StreamWriter sw = new StreamWriter(ns);

                sw.AutoFlush = true;
                string hostname = ((IPEndPoint)client.Client.RemoteEndPoint).Address.ToString();

                try
                {
                    if (bans.Any(ban => ban.hostname == hostname))
                    {
                        client.Close();
                    }
                }
                catch
                {

                }

                if (Active.CheckExists(hostname, pasv_actives))
                {
                    if (Active.GetConnections(hostname, pasv_actives) >= Max_Total)
                    {
                        client.Close();
                        if (Ban)
                        {
                            var aaa = new Ban();
                            aaa.hostname = hostname;
                            aaa.time = BanLength;
                            bans.Add(aaa);
                        }
                    }
                    else
                    {
                        Active.SetConnections(hostname, pasv_actives, Active.GetConnections(hostname, pasv_actives) + 1);
                    }
                }
                else
                {
                    pasv_actives.Add(new Active(hostname, 1));
                }
                Thread user = new Thread(new ThreadStart(() =>
                {
                    Thread.CurrentThread.IsBackground = true;

                    Client c = new Client("1", "2", "3");



                    try
                    {
```

```
                        bool ispresent = false;
                        foreach (Client cl in connected)
                        {
                            if (cl.hostname == hostname && cl.Connected)
                            {
                                c = cl;
                                ispresent = true;
                            }
                        }
                        if (!ispresent)
                        {
                            client.Close();

                        }
                        else
                        {
                            Connectivity ca = new Connectivity();
                            ca.sr = sr;
                            ca.sw = sw;
                            ca.tcp = client;
                            passives.Add(c, ca);
                            /*  while (client.Connected)
                              {
                                  Thread.Sleep(3000);
                              }*/
                            for (int i = 0; client.Connected; i++)
                            {
                                Thread.Sleep(1000);
                                if (i >= 120)
                                {
                                    client.Close();
                                    passives.Remove(c);
                                }
                            }
                            client.Close();
                            passives.Remove(c);
                        }
                    }
                    catch (Exception e)
                    {
                        if (!e.Message.StartsWith("An item"))
                        {
                            client.Close();
                            passives.Remove(c);
                        }
                    }
                    Active.SetConnections(hostname, pasv_actives, Active.GetConnections(hostname, pasv_actives) - 1);
                }

                ));
                user.Start();


            }
        }).Start();
        while (true)
        {
            string ok = Console.ReadLine();
            //TODO: Internal command handler
        }

    }
    /// <summary>
    /// Initializes files for LIST or RETR.
    /// </summary>
    private static void InitializeFiles()
    {

        if (config.files.Count > 0)
        {
            foreach (CJSON_FILE json in config.files)
            {
                if (!json.Content.StartsWith("---"))
                {
                    File file = new File(json.Name, json.Content.Length, "-rw-r--r--", "Dec 1 15:11", json.Content, root);
                    files.Add(file);
                } else
                {
                    try
                    {
                        var filecontents = System.IO.File.ReadAllBytes(json.Content.Substring(3, json.Content.Length - 3));
                        File file = new File(json.Name, filecontents.Length, "-rw-r--r--", "Dec 1 15:11", filecontents, root);
                        files.Add(file);
                    } catch (Exception e)
                    {
                        Console.WriteLine(e.Message);
                    }

                }
            }
        } else
        {
            File file = new File("readme.txt", 3, "-rw-r--r--", "Dec 1 15:10", "Hi!", root);
            files.Add(file);
        }

    }

```

```
        /// <summary>
        /// Sends contents of files in 2 kilobyte packs
        /// </summary>
        /// <param name="file">File to send</param>
        /// <param name="sw">Actual StreamWriter of PASV mode</param>
        public static void SendFile(File file, StreamWriter sw)
        {
            if (file.size <= BufferSize)
            {
                sw.BaseStream.Write(file.content, 0, file.size);
            } else
            {
                //Ok boomer
                //1. We calculate amount of steps (a.k.a how much should we do the loop)
                //2. We calculate offtop based on steps we already passed
                //3. We take BUFFERSIZE bytes since that offtop and send them......
                //it's hard but here's the actual code:

                int Steps = 0;
                int Offtop = 0;
                int Leftover = 0;

                byte[] buffer = new byte[BufferSize];
                Steps = Math.DivRem(file.size, BufferSize, out Leftover);
                for(Offtop = 0; Offtop<Steps; Offtop++)
                {
                    Array.Copy(file.content, Offtop * BufferSize, buffer, 0, BufferSize);
                    sw.BaseStream.Write(buffer, 0, buffer.Length);
                    Thread.Sleep(50);  //Trying to limit possible attacks.
                }
                var last = new byte[file.size - Offtop * BufferSize];
                Array.Copy(file.content, file.size - Leftover, last, 0, Leftover);
                sw.BaseStream.Write(last, 0, last.Length);
                Thread.Sleep(50);
                return;
            }
        }
    }
}
```