Napier University

Botnet Analysis and Prototype Intrusion Detection System

An analysis of a provided botnet and design and implementation of a prototype IDS and firewall

40435990
12-17-2021

Contents

Introduction

Botnets are a major security threat and widespread and difficult to detect. They can infect the majority of devices that can communicate within a network. Botnets can control, disable, and wipe devices leading to failures in network infrastructure that costs administrators' vast amounts of money. This report will discuss the testing and analysis of a particular botnet controller and its agent within a small network. How it operates and what it does will be discussed, analysed, and tested with the intention of building a prototype IDS system and firewall to counteract/prevent the infection of the botnet into/within the network.

Research

A botnet is malware that has infected a host computer where it is instructed by a bot controller to perform malicious activities that steal, harm, or disable a network or host. Botnets differentiate themselves from other malware as they employ the use of a command-and-control channel to receive commands and update software (Basil, Abdulmuneem, Jalal, & Saleh, 2018).

A botnet can be classified into two separate types each with a different communications method. One type uses Internet Relay Chat (IRC) as a communication channel and the other uses Hyper Text Transfer Protocol (HTTP). Using HTTP for a botnet can mask botnet traffic due to the large volumes of HTTP packets used in web browsing and other services. This has led to HTTP being chosen more frequently over its counterpart (IRC) when creating botnets. In 2007 ICR botnets were estimated to be around 36% (Jianwei, Thorsten, Xinhui, Jinpeng, & Wei, 2007) of all botnets (Using the port 6667 this number is most definitely decreased as new methods emerge such as Peer to Peer (P2P) botnets emerge.

To detect the botnet one of two IDS methods (Network-based detection or host-based detection) are used. Network-based detection is the process of analysing network traffic that is suspected to be host to botnet activities in the hopes of finding irregularities and patterns that the botnet may produce. This is the preferred method of botnet detection over host-based detection (Baruaha, 2018) as it is harder for a botnet to hide its presence, attempting to do so has the potential to create more network traffic that might be flagged by a thorough IDS. Host-based detection involves analysing the behaviour and state of a computer system and analysing network traffic going in and out of the host itself. This method struggles as visibility is limited to a single host and consumes host resources impacting the performance of other services (Thomas M. & Patrick J., 2014).

Network

As a testbed, 4 virtual machines were used to create a small network (Figure 1). It included a PFSense firewall that connected a public network to "DMZ" and Private networks. The private network had an Ubuntu machine that was used to configure the PFSense firewall. The "DMZ" contained a windows machine running the botnet controller and a kali Linux machine running the botnet agent and Wireshark.

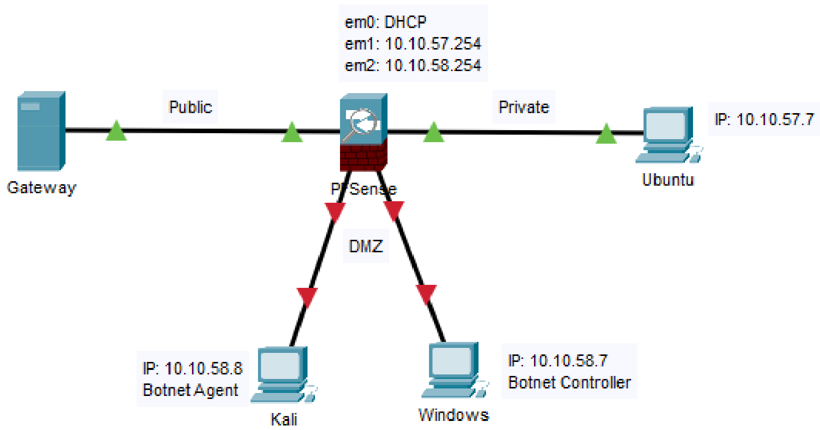


Figure 1

Connectivity

The PFSense firewall rules were configured to allow all traffic to traverse between all parts of the network (Figure 2). This was to ensure the botnet could communicate with no barriers this meant the botnet would provide as much data as possible for analysis ensuring that when an IDS and firewall were implemented it would cover all necessary areas of defence and detection. The connection between devices and firewall was tested using ICMP pings to ensure all necessary parts of the network were reachable (Figure 3). Proof of internet connection in appendix (Figure 31).

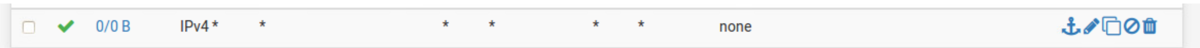


Figure 2

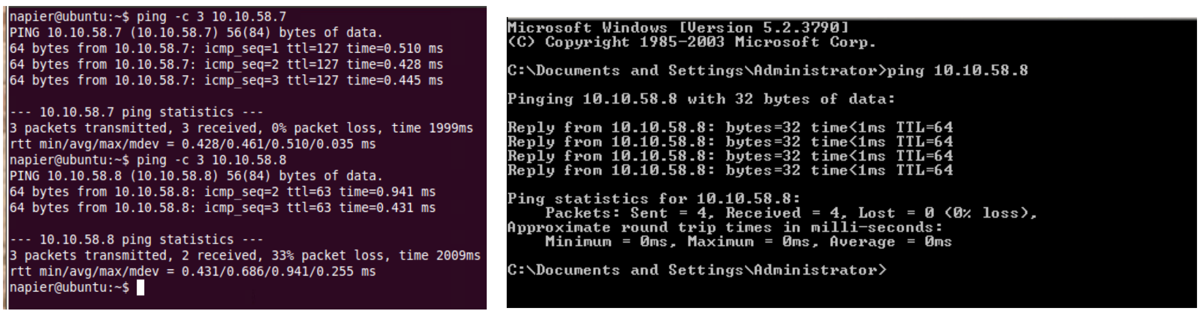


Figure 3

Wireshark Analysis

Using Wireshark, traffic was captured on the wired interface on the kali machine (Eth0). The bot agent was then run 4 consecutive times trying to capture any differences in botnet behaviour between runs (bots). According to the task specification, only traffic between the bot and the controller is required, other traffic such as router advertisements must be filtered. This issue was solved by filtering the traffic to only display packets traversing between the kali and windows machines.

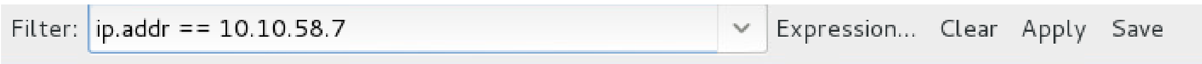
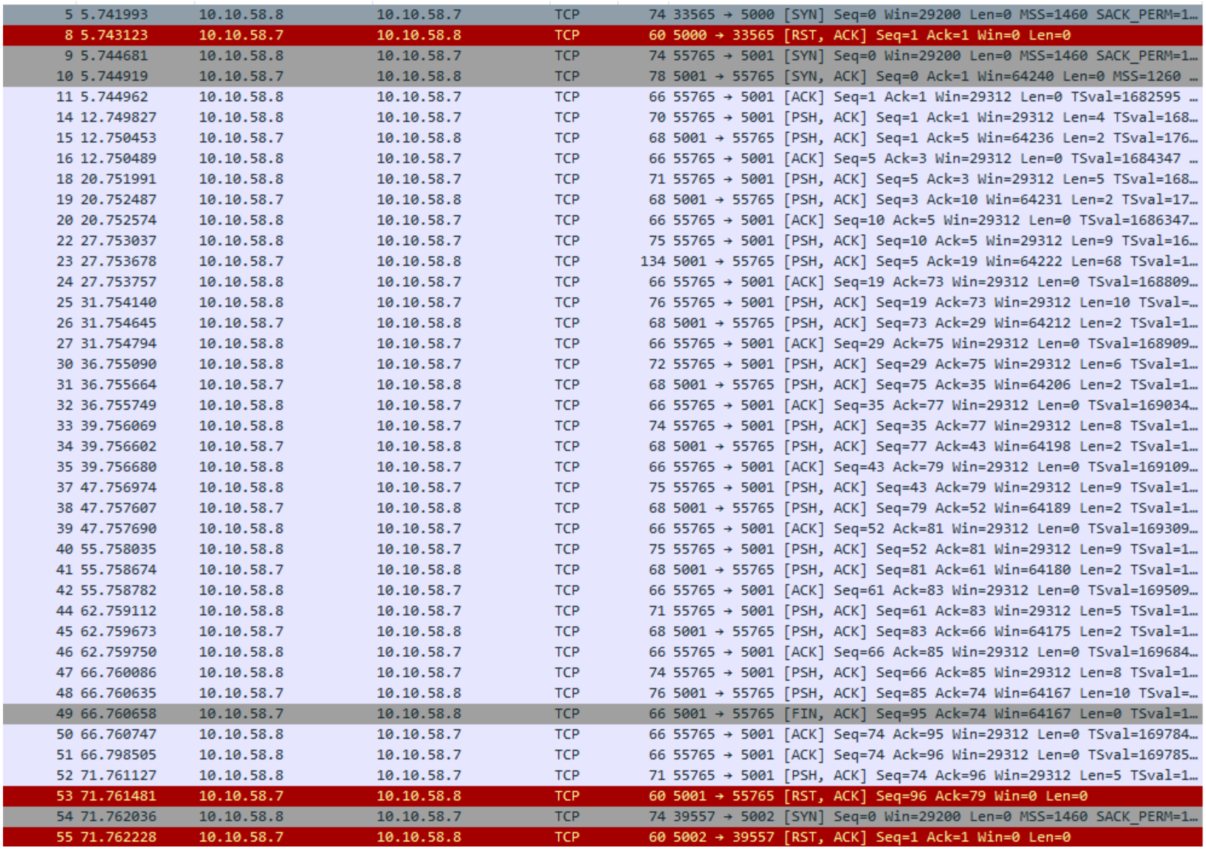


Figure 4

Figure 5

To do this the rules ip.addr == 10.10.58.7(Figure 4)or ip.addr == 10.10.58.8 (Figure 5) were applied. The traffic left after filtering is a TCP connection between the kali and windows machines (Figure 6). The machines have no other connections between them meaning this traffic it is almost certainly from the botnet.



5	5.741993	10.10.58.8	10.10.58.7	TCP	74	33565 → 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1…
8	5.743123	10.10.58.7	10.10.58.8	TCP	60	5000 → 33565 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	5.744681	10.10.58.8	10.10.58.7	TCP	74	55765 → 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1…
10	5.744919	10.10.58.7	10.10.58.8	TCP	78	5001 → 55765 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1260 …
11	5.744962	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1682595 …
14	12.749827	10.10.58.8	10.10.58.7	TCP	70	55765 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=4 TSval=168…
15	12.750453	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=1 Ack=5 Win=64236 Len=2 TSval=176…
16	12.750489	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=5 Ack=3 Win=29312 Len=0 TSval=1684347 …
18	20.751991	10.10.58.8	10.10.58.7	TCP	71	55765 → 5001 [PSH, ACK] Seq=5 Ack=3 Win=29312 Len=5 TSval=168…
19	20.752487	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=3 Ack=10 Win=64231 Len=2 TSval=17…
20	20.752574	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=10 Ack=5 Win=29312 Len=0 TSval=1686347…
22	27.753037	10.10.58.8	10.10.58.7	TCP	75	55765 → 5001 [PSH, ACK] Seq=10 Ack=5 Win=29312 Len=9 TSval=16…
23	27.753678	10.10.58.7	10.10.58.8	TCP	134	5001 → 55765 [PSH, ACK] Seq=5 Ack=19 Win=64222 Len=68 TSval=1…
24	27.753757	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=19 Ack=73 Win=29312 Len=0 TSval=168809…
25	31.754140	10.10.58.8	10.10.58.7	TCP	76	55765 → 5001 [PSH, ACK] Seq=19 Ack=73 Win=29312 Len=10 TSval=…
26	31.754645	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=73 Ack=29 Win=64212 Len=2 TSval=1…
27	31.754794	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=29 Ack=75 Win=29312 Len=0 TSval=168909…
30	36.755090	10.10.58.8	10.10.58.7	TCP	72	55765 → 5001 [PSH, ACK] Seq=29 Ack=75 Win=29312 Len=6 TSval=1…
31	36.755664	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=75 Ack=35 Win=64206 Len=2 TSval=1…
32	36.755749	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=35 Ack=77 Win=29312 Len=0 TSval=169034…
33	39.756069	10.10.58.8	10.10.58.7	TCP	74	55765 → 5001 [PSH, ACK] Seq=35 Ack=77 Win=29312 Len=8 TSval=1…
34	39.756602	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=77 Ack=43 Win=64198 Len=2 TSval=1…
35	39.756680	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=43 Ack=79 Win=29312 Len=0 TSval=169109…
37	47.756974	10.10.58.8	10.10.58.7	TCP	75	55765 → 5001 [PSH, ACK] Seq=43 Ack=79 Win=29312 Len=9 TSval=1…
38	47.757607	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=79 Ack=52 Win=64189 Len=2 TSval=1…
39	47.757690	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=52 Ack=81 Win=29312 Len=0 TSval=169309…
40	55.758035	10.10.58.8	10.10.58.7	TCP	75	55765 → 5001 [PSH, ACK] Seq=52 Ack=81 Win=29312 Len=9 TSval=1…
41	55.758674	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=81 Ack=61 Win=64180 Len=2 TSval=1…
42	55.758782	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=61 Ack=83 Win=29312 Len=0 TSval=169509…
44	62.759112	10.10.58.8	10.10.58.7	TCP	71	55765 → 5001 [PSH, ACK] Seq=61 Ack=83 Win=29312 Len=5 TSval=1…
45	62.759673	10.10.58.7	10.10.58.8	TCP	68	5001 → 55765 [PSH, ACK] Seq=83 Ack=66 Win=64175 Len=2 TSval=1…
46	62.759750	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=66 Ack=85 Win=29312 Len=0 TSval=169684…
47	66.760086	10.10.58.8	10.10.58.7	TCP	74	55765 → 5001 [PSH, ACK] Seq=66 Ack=85 Win=29312 Len=8 TSval=1…
48	66.760635	10.10.58.7	10.10.58.8	TCP	76	5001 → 55765 [PSH, ACK] Seq=85 Ack=74 Win=64167 Len=10 TSval=…
49	66.760658	10.10.58.7	10.10.58.8	TCP	66	5001 → 55765 [FIN, ACK] Seq=95 Ack=74 Win=64167 Len=0 TSval=1…
50	66.760747	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=74 Ack=95 Win=29312 Len=0 TSval=169784…
51	66.798505	10.10.58.8	10.10.58.7	TCP	66	55765 → 5001 [ACK] Seq=74 Ack=96 Win=29312 Len=0 TSval=169785…
52	71.761127	10.10.58.8	10.10.58.7	TCP	71	55765 → 5001 [PSH, ACK] Seq=74 Ack=96 Win=29312 Len=5 TSval=1…
53	71.761481	10.10.58.7	10.10.58.8	TCP	60	5001 → 55765 [RST, ACK] Seq=96 Ack=79 Win=0 Len=0
54	71.762036	10.10.58.8	10.10.58.7	TCP	74	39557 → 5002 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1…
55	71.762228	10.10.58.7	10.10.58.8	TCP	60	5002 → 39557 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 6

Ports

Initial analysis of the remaining traffic showed an attempted connection (Bot 1), through ports 36697 and 5001 (commplex-link) and a successful connection through ports 55124 and 5001 [Other attempted/successful connections, Table 1]. The largely unused and uncommon port numbers such as 55765 and 35911 indicate the bot agent was created to exploit ports that are less likely to be secured (be open) on a network.

Table 1

Bot	Source Port	Destination Port	Successful?
1	33565	5000	No
1	55765	5001	Yes
2	35911	5000	No
2	57087	5001	Yes
3	60839	5000	No
3	55240	5001	Yes
4	54884	5000	No
4	52705	5001	Yes

TCP Stream

Once the bot agent had successfully connected to its controller it began transmitting TCP packets. Some packets had data containing hexadecimal strings. Bot 1 contained 21 such packets, Bot 2, 3 packets, Bot 3, 9 packets and Bot 4, 9 packets. The strings varied in length from 2 to 68 bytes (Figure 7, Figure 8).

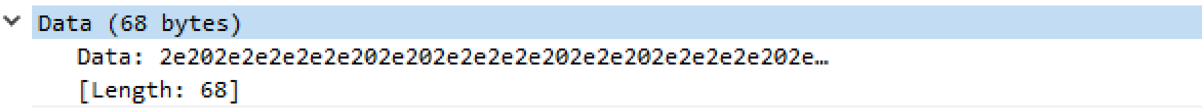


Figure 7

Figure 8

To help with analysis the stream was selected using a filter in Wireshark to show TCP streams (Figure 9).

Figure 9

Hexadecimal Conversion

The strings when converted to ASCII contained words such as generate, failover and capture [Table 2]. The number and sequence of commands varied between bots, but the content of the commands remained unchanged.

Table 2

Bot	Hexadecimal	ASCII
1	6765740a	get
1	6c6f6f700a	loop
1	67656e65726174650a	generate
1	2e202e2e2e2e2e202e202e2e2e2e202e2e 202e2e2e2e202e2e2e202e2e2e2e202e202e2 e202e2e2e2e2e202e202e2e2e2e2 02e2e2e202e2e202e2e202e2e202e2e2e0d0a	. ...
3	2e2e2e202e2e2e202e2e2e202e2e2e2e2e202 e2e2e202e2e202e202e2e2e2e202e2e202e2e 2e2e202e2e2e202e2e2e2e202e2e202e2e0d0a	... ...
1	6b656570616c6976650a	keepalive
1	68656c6c6f0a	hello
1	636170747572650a	capture
1	6661696c6f7665720a	failover
1	74616b65646f776e0a	takedown
1	746573740a	test
1	676f6f646279650a	goodbye
1	736c6565702e2e2e0d0a	sleep...
1	636f64650a	Code
1	0d0a	Newline (Literal)

The bot-agent was not observed sending any data other than the words shown above however packets containing only dots spaced out like morse code were found on two separate bot communications (1 and 3). This however when translated was nothing more than random letters and numbers.

Internet Relay Chat (IRC)

Looking at the characteristics of IRC (Single line, plaintext, client-to-server commands) and comparing them to the results after the Hexadecimal conversion it is most likely the botnet uses IRC as its communication method. The IRC assigned port is 6667 but can run on other ports with root privileges which makes it hard to definitively say if the botnet is using IRC communications.

End Transmission

After the initial TCP stream, a service request is made from the bot-agent to port 5002 on the controller's machine (Figure 10). This port is particularly interesting as it has been reported to be used by other malware such as W32.Spybot.IVQ (SpeedGuide Inc, 2021) and Linux Rootkit IV (Internet Storm Center, 2021). There is no service available for the bot-agent on that port as the controller responds with a [RST, ACK] packet.

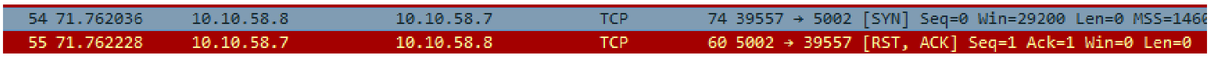


Figure 10

Snort IDS

Based on the analysis of the bot-agent and controller a rudimentary IDS can be built. The IDS software that will be used is called SNORT. SNORT is an open-source intrusion detection system (IDS). The software utilises three main modes Sniffer, Packet Logger, and Network IDS this allows the software to analyse traffic in real-time and packet log on IP enabled networks. SNORT can be used to detect a wide range of attacks such as buffer overflows, stealth port scans, and semantic URL attacks (CISCO, 2021).

SNORT can be used as a signature detection IDS this means that SNORT uses a library of signatures to compare to network traffic looking for matches which it can then log. An advantage of SNORT being signature-based is known attacks are clearly defined (Shah & Singh, 2012). This means the system admin can be directed by SNORT to specific traffic/activity reducing the effort required by the admin when trying to find the source of the attack so it can be prevented. However, since this works based on known attacks, SNORT when using signature-based methods struggles with unknown or new signatures. This can be mitigated by frequently updating the signature library or implementing SNORT other anomaly-based features or alongside another IDS or IDPS specialised in anomaly-based detection.

Snort Rules

To allow SNORT to detect, log or create alerts it must know what it is looking for. SNORT uses rules to define what traffic the system administrator should be alerted about, what traffic should be logged and what traffic should be blocked. SNORT can look at a wide range of traffic but based on the botnet analysis the SNORT rules for the network IDS will focus on TCP ports, TCP connections, and signature detection based on known botnet content/commands.

Testing of created rules consisted of running of an individual rule and the PCAP file used in botnet analysis using the command "*snort -r pcap3.pcapng -c rule.rules".* This produced a SNORT report with the number of packets found by SNORT shown by the number of alerts. This could then be manually compared to the PCAP file using Wireshark. This method provides useful information that can show the number of false positives and/or missed packets this helps to produce a rule that effectively does what it is designed to do.

Snort Rules Continued

Source Ports (Controller Ports)

From the botnet analysis, it was observed that the bot controller would use the ports 4997, 4998, 4999, 5000, 5001, 5002, 5003, 5004, and 5005 when communicating with the bot-agent. Ports 4997-4999 and 5003-5005 were added to account for errors in analysis. A rule was created for when traffic that had a source-destination from the suspected ports was detected, the IDS would flag the packet (Figure 11).

```
#alerts for traffic sourced from suspected ports
alert tcp any [5000,5001,5002,5003,5004,5005] -> any any (msg:"Known Malicious Port Activity"; sid:10001;)
```

Figure 11

Source Ports - Testing

```
Action Stats:
     Alerts:           39 ( 24.375%)
     Logged:           39 ( 24.375%)
     Passed:            0 (  0.000%)
```

Figure 12

```
[**] [1:1000001:1] Known Malicious Port Activity [**]
[Priority: 0]
12/01-17:12:43.866591 10.10.58.7:5000 -> 10.10.58.8:36697
TCP TTL:128 TOS:0x0 ID:3883 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0x58542A5F  Win: 0x0  TcpLen: 20
```

Figure 13

Destination Ports (Bot-agent Ports)

The analysis showed the controller preferred to use ports above port 33000 - the lowest recorded port used by the bot being 33565. To provide a suitable margin for error the next rule was created to log activity on ports higher than 30000 (Figure 14). The rule only logs traffic as it was considered unnecessary to create an alert since (although uncommon) legitimate services do use ports above 30000 this would unintentionally create many false positive alerts.

```
#alerts for traffic on higher ports
log tcp any 30000:65535 -> any any (msg:"Possible Un-used Port Activity"; sid:10002;)
```

Figure 14

Destination Ports - Testing

```
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:           39 ( 24.375%)
     Passed:            0 (  0.000%)
```

Figure 15

No.	Time	Source	Destination	Protocol	Lengtl	Info
1	0.000000	10.10.58.7	10.10.58.8	TCP	60	commplex-main > 33565 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2	0.001796	10.10.58.7	10.10.58.8	TCP	78	commplex-link > 55765 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1260 WS=1 T
3	7.007330	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=1 Ack=5 Win=64236 Len=2 TSval=176458 TS
4	15.009364	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=3 Ack=10 Win=64231 Len=2 TSval=176537 T
5	22.010555	10.10.58.7	10.10.58.8	TCP	134	commplex-link > 55765 [PSH, ACK] Seq=5 Ack=19 Win=64222 Len=68 TSval=176607
6	26.011522	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=73 Ack=29 Win=64212 Len=2 TSval=176648
7	31.012541	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=75 Ack=35 Win=64206 Len=2 TSval=176697
8	34.013479	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=77 Ack=43 Win=64198 Len=2 TSval=176727
9	42.014484	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=79 Ack=52 Win=64189 Len=2 TSval=176808
10	50.015551	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=81 Ack=61 Win=64180 Len=2 TSval=176887
11	57.016550	10.10.58.7	10.10.58.8	TCP	68	commplex-link > 55765 [PSH, ACK] Seq=83 Ack=66 Win=64175 Len=2 TSval=176957
12	61.017512	10.10.58.7	10.10.58.8	TCP	76	commplex-link > 55765 [PSH, ACK] Seq=85 Ack=74 Win=64167 Len=10 TSval=176998
13	61.017535	10.10.58.7	10.10.58.8	TCP	66	commplex-link > 55765 [FIN, ACK] Seq=95 Ack=74 Win=64167 Len=0 TSval=176998
14	66.018358	10.10.58.7	10.10.58.8	TCP	60	commplex-link > 55765 [RST, ACK] Seq=96 Ack=79 Win=0 Len=0
15	66.019105	10.10.58.7	10.10.58.8	TCP	60	rfe > 39557 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 16

Attempted Connections

Since the bot-agent would routinely attempt to connect to ports that refused its connection a rule that flagged attempted connections was created (Figure 17). To make the rule more specific and reduce false positives, it was limited to the ports the bot-agent frequented. The rule matched the RST flag when alerting as this meant there was a TCP connection that had been terminated/refused.

```
#Alerts for failed connections to ports
alert tcp any any -> any 30000:65535 (flags:*R; msg:"Attempted But Blocked Connection"; sid:10003;)
```

Figure 17

Attempted Connection - Testing

```
Action Stats:
     Alerts:           12 (  7.500%)
     Logged:           12 (  7.500%)
     Passed:            0 (  0.000%)
```

Figure 18

```
[**] [1:10003:0] Attempted But Blocked Connection [**]
[Priority: 0]
11/30-15:32:06.397572 10.10.58.7:5000 -> 10.10.58.8:33565
TCP TTL:128 TOS:0x0 ID:985 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0xA0558C8B  Win: 0x0  TcpLen: 20
```

Figure 19

Live Testing

Once the rules were created and tested it was necessary to ensure they properly functioned in a real environment. To do this SNORT was run to capture traffic on the KALI machine's eth0 interface where it would apply the rules to the traffic entering and exiting the machine. The botnet was then run twice to ensure it would capture a large variety of botnet traffic.

Results - Source Ports/Attempted Connections

Shown below (Figure 20) are alerts produced from the Source Ports and Attempted Connections meaning it can be concluded that those rules work as intended producing the expected output.

```
[**] [1:10003:0] Attempted But Blocked Connection [**]
[Priority: 0]
12/10-16:36:10.510825 10.10.58.7:5000 -> 10.10.58.8:53952
TCP TTL:128 TOS:0x0 ID:2268 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0x98007989  Win: 0x0  TcpLen: 20

[**] [1:10001:0] Known Malicious Port Activity [**]
[Priority: 0]
12/10-16:36:10.510825 10.10.58.7:5000 -> 10.10.58.8:53952
TCP TTL:128 TOS:0x0 ID:2268 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0x98007989  Win: 0x0  TcpLen: 20

[**] [1:10003:0] Attempted But Blocked Connection [**]
[Priority: 0]
12/10-16:36:10.512067 10.10.58.7:5001 -> 10.10.58.8:47943
TCP TTL:128 TOS:0x0 ID:2269 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0xDB7F2F70  Win: 0x0  TcpLen: 20

[**] [1:10001:0] Known Malicious Port Activity [**]
[Priority: 0]
12/10-16:36:10.512067 10.10.58.7:5001 -> 10.10.58.8:47943
TCP TTL:128 TOS:0x0 ID:2269 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0xDB7F2F70  Win: 0x0  TcpLen: 20

[**] [1:10003:0] Attempted But Blocked Connection [**]
[Priority: 0]
12/10-16:36:10.512572 10.10.58.7:5002 -> 10.10.58.8:58130
TCP TTL:128 TOS:0x0 ID:2270 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0xC6EE07D7  Win: 0x0  TcpLen: 20
```

Figure 20

Results - Destination Ports

Show below (Figure 21) is a sample of the traffic logged by the destination ports rule. This rule produces a lot of false positives however it is necessary for tracking traffic that is going through uncommon ports that the botnet was observed using.

1	0.000000	10.10.58.7	10.10.58.8	TCP	60 commplex-main > 45482 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2	0.001604	10.10.58.7	10.10.58.8	TCP	60 commplex-link > 40363 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.002100	10.10.58.7	10.10.58.8	TCP	60 rfe > 59515 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4	0.002588	10.10.58.7	10.10.58.8	TCP	60 fmpro-internal > 60810 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.003070	10.10.58.7	10.10.58.8	TCP	78 avt-profile-1 > 50938 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1260 WS=1 TS
6	0.004133	10.10.58.7	10.10.58.8	TCP	78 avt-profile-1 > 41418 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1260 WS=1 TS
7	4.011895	10.10.58.7	10.10.58.8	TCP	134 avt-profile-1 > 41418 [PSH, ACK] Seq=1 Ack=43 Win=64198 Len=68 TSval=57184 TSe
8	4.013329	10.10.58.7	10.10.58.8	TCP	66 avt-profile-1 > 41418 [ACK] Seq=69 Ack=44 Win=64198 Len=0 TSval=57184 TSecr=2
9	4.015392	10.10.58.7	10.10.58.8	TCP	66 avt-profile-1 > 50938 [ACK] Seq=1 Ack=2 Win=64240 Len=0 TSval=57184 TSecr=238
10	8.738591	52.111.236.8	10.10.58.8	TLSv1.2	86 Application Data

Figure 21

Firewall

For the PFSense firewall, the rules to block botnet traffic were based upon what was discovered in the analysis. This meant blocking ports such as the source and destination ports that the botnet frequents and stopping communication effectively crippling the botnet. The firewall must also allow for normal internet access according to the specification (Figure 32).

Firewall Rules - IPv4 Traffic

To ensure botnet communication was limited as much as possible traffic was limited to only TCP/UDP and ICMP. This was a preventative measure to stop communications using other IPv4 protocols. This was implemented using 3 rules, one that rejected all IPv4 traffic (Figure 22), one that passed TCP/UDP traffic (Figure 23), and a final rule that allowed ICMP traffic (Figure 24). The IPv4 rule was configured to reject meaning a response was sent back to the source instead of quietly dropping the packet as with block.

0/33 KiB	IPv4 *	*	*	*	*	*	none		

Figure 22

0/656 B	IPv4 TCP/UDP	*	*	*	*	*	none		

Figure 23

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
0/0 B	IPv4 ICMP	*	*	*	*	*	none			

Figure 24

Firewall Rules - Source Ports

To stop the botnet controller from sending packets to the bot-agent the source ports 4997-5005 were closed. This was implemented through two rules on all interfaces stating that TCP/UDP traffic going through the firewall and from the specified ports was to be blocked and then logged for administrative purposes (Figure 25).

0/0 B	IPv4 TCP/UDP	*	4997 - 5005	*	*	*	none
0/0 B	IPv4 TCP/UDP	*	*	*	4997 - 5005	*	none

Figure 25

Firewall Rules - Destination Ports

To stop the bot-agent from communicating with its controller packets with a destination between 30000 and 65635 were closed. This was implemented through two rules stating that traffic going through the firewall and to the ports specified should be rejected and logged (Figure 26).

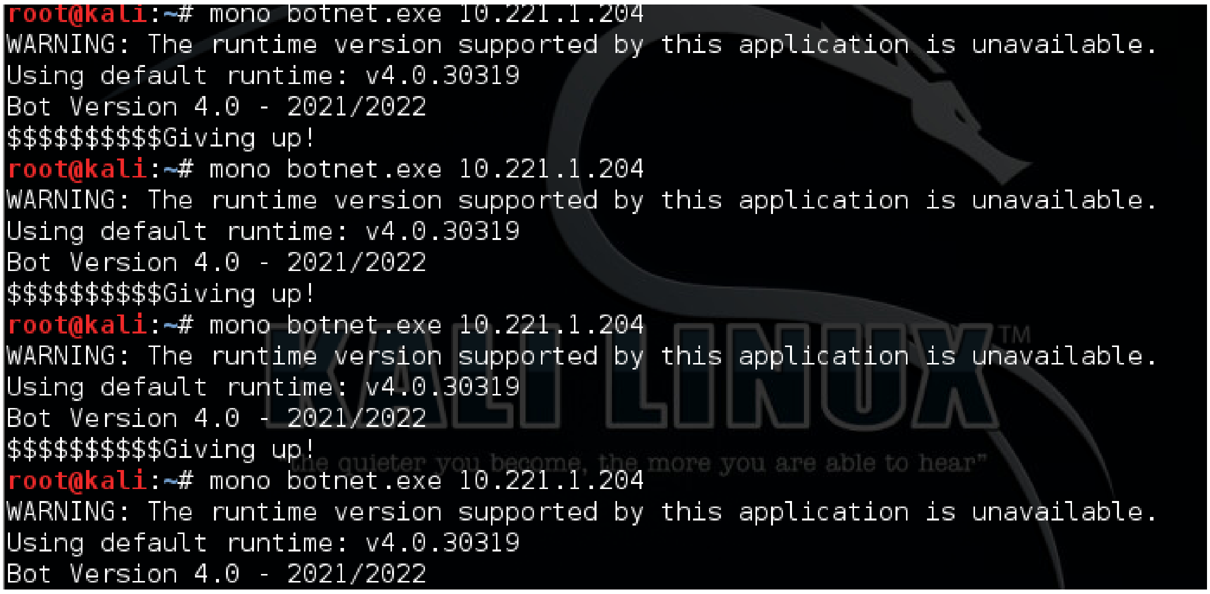
0/0 B	IPv4 TCP	*	30000 - 65535	*	*	*	none
0/0 B	IPv4 TCP	*	*	*	30000 - 65535	*	none

Figure 26

Firewall Rules - Testing

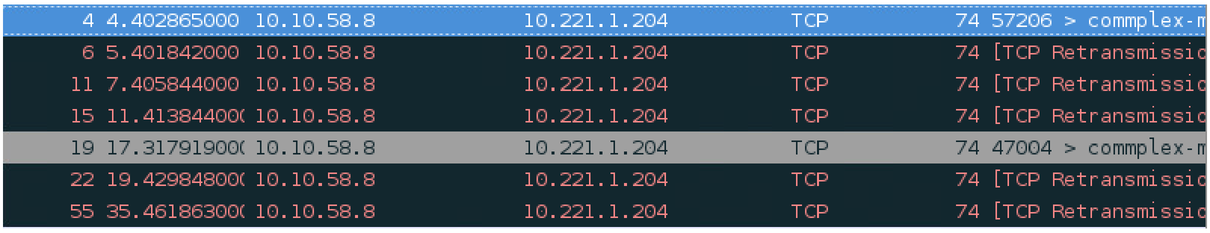
To simulate a real-life scenario the botnet controller was placed out with the DMZ and private networks. To do this a kali machine on the WAN network was added. Since the WAN network uses DHCP there was no need to configure a static Ip address.

The firewall is intended to stop communications between the controller and the bot-agent thus stopping the botnet from gathering and stealing sensitive data. To test this the botnet was run 4 times (Figure 27) with traffic from the botnet being captured on both machines using Wireshark (Figure 28).



```
root@kali:~# mono botnet.exe 10.221.1.204
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$$$$$$$$$$Giving up!
root@kali:~# mono botnet.exe 10.221.1.204
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$$$$$$$$$$Giving up!
root@kali:~# mono botnet.exe 10.221.1.204
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$$$$$$$$$$Giving up!
root@kali:~# mono botnet.exe 10.221.1.204
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
```

Figure 27



4	4.402865000	10.10.58.8	10.221.1.204	TCP	74	57206 > commplex-m
6	5.401842000	10.10.58.8	10.221.1.204	TCP	74	[TCP Retransmissio
11	7.405844000	10.10.58.8	10.221.1.204	TCP	74	[TCP Retransmissio
15	11.41384400(	10.10.58.8	10.221.1.204	TCP	74	[TCP Retransmissio
19	17.31791900(	10.10.58.8	10.221.1.204	TCP	74	47004 > commplex-m
22	19.42984800(	10.10.58.8	10.221.1.204	TCP	74	[TCP Retransmissio
55	35.46186300(	10.10.58.8	10.221.1.204	TCP	74	[TCP Retransmissio

Figure 28

Figure 28 (above) shows the bot-agent attempting to communicate with its controller but failing and sending a re-transmission this pattern continues until a final transmission is sent and the bot-agent gives up.

Improvements

While testing the firewall configuration it was noted that the bot controller used ports up to 5009. This was higher than the 5005 implemented in both the firewall configuration and SNORT IDS however since ports 30000 to 65535 were blocked the issue went unknown during testing. Even though the firewall was preventing the botnet from operating the firewall and IDS were reconfigured to alert/block up to port 5015 (Figure 29, Figure 30).

```
#alerts for traffic sourced from suspected ports
alert tcp any [5000,5001,5002,5003,5004,5005,5006,5007,5008,5009,5010,5011,5012,5013,5014,5015] -> any any (msg:"Known Malicious Port Activity"; sid:10001;)
```

Figure 29

0/0 B	IPv4 TCP/UDP	*	4997 - 5015	*	*	*	none	Suspected Botnet Ports
0/1 KiB	IPv4 TCP/UDP	*	*	*	4997 - 5015	*	none	

Figure 30

EndNote

This report was completed according to the CSN09112 - Network Security and Cryptography coursework specification authored by Bill Buchanan. This report meets the specification by analysing a provided botnet, creating an IDS prototype, and implementing and testing a firewall defence.

Appendix

Below is the ubuntu machine connecting to the NHS website during analysis stage

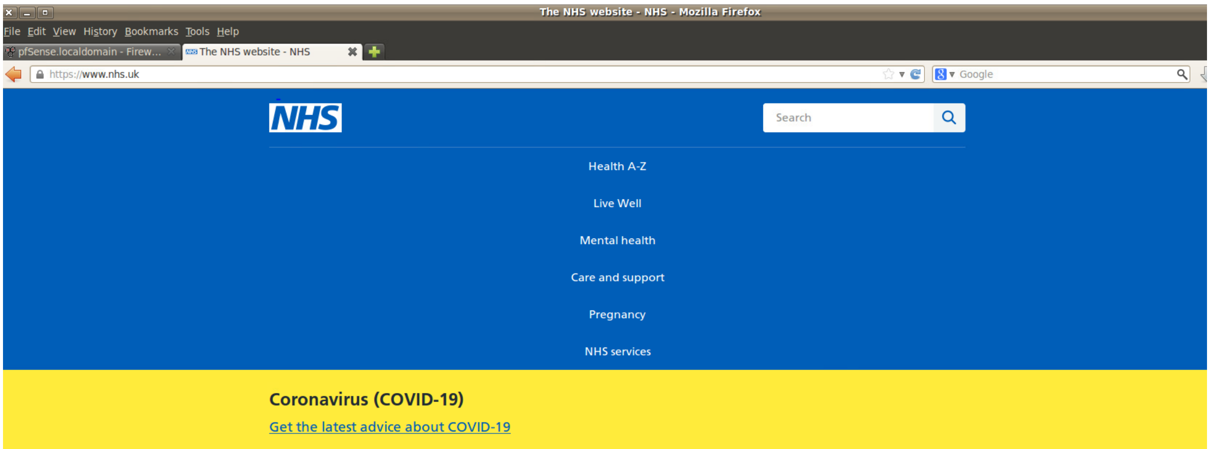


Figure 31

Below is the Ubuntu machine connecting to the .gov website during the firewall testing stage

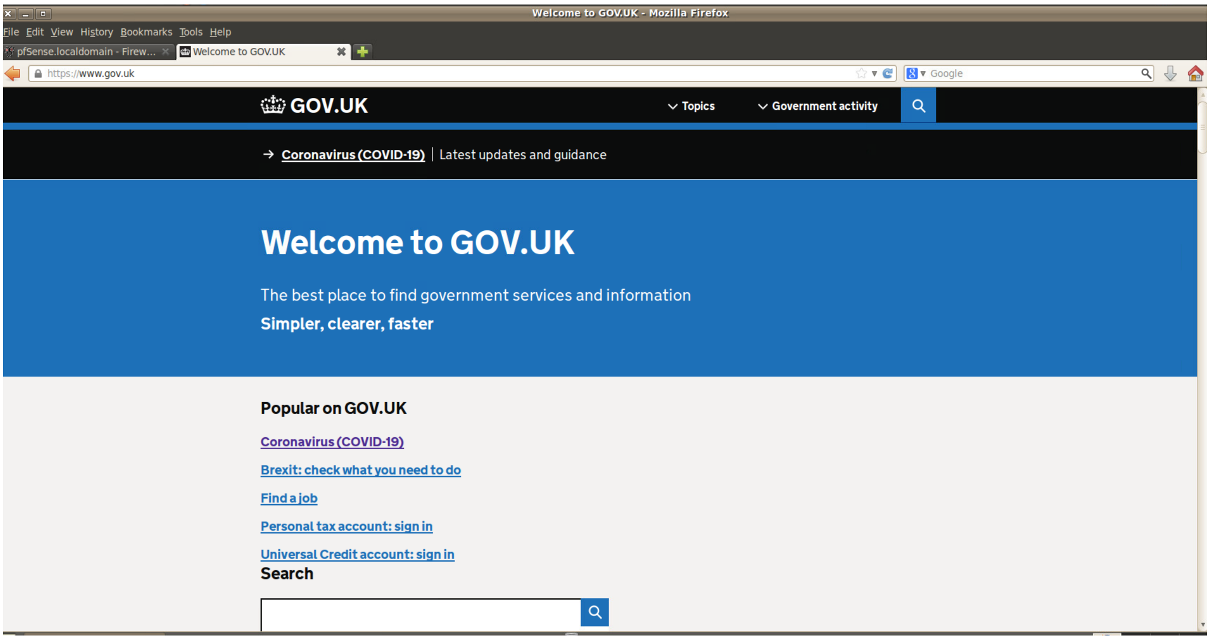


Figure 32

Works Cited

Baruaha, S. (2018). Botnet Detection: Analysis of Various Techniques. *International Conference on Computational Intelligence & IoT (ICCIIoT)* (p. 7). AGARTALA: SSRN.

Basil, A., Abdulmuneem, B., Jalal, A.-M., & Saleh, A. (2018). Analysis of P2P, IRC and HTTP traffic for botnets detection. *Peer-to-Peer Network Applications*.

CISCO. (2021). *SNORT FAQ*. Retrieved from SNORT: https://www.snort.org/faq/what-is-snort

Internet Storm Center. (2021). *Port 5002 (tcp/udp) Attack Activity*. Retrieved from Internet Storm Center: https://isc.sans.edu/port.html?port=5002

Jianwei, Z., Thorsten, H., Xinhui, H., Jinpeng, G., & Wei, Z. (2007). *Characterizing the IRC-based Botnet Phenomenon.* Beijing, Mannheim: Peking University Institute of Computer Science and Technology, University of Mannheim Laboratory for Dependable Distributed Systems.

Shah, S. N., & Singh, P. (2012). Signature-Based Network Intrusion Detection System. *International Journal of Engineering Research & Technology (IJERT)*.

SpeedGuide Inc. (2021). *Port 5002 Details*. Retrieved from SpeedGuide.net: https://www.speedguide.net/port.php?port=5002

Thomas M., C., & Patrick J., W. (2014). Chapter 3 - Guarding Against Network Intrusions. In J. R. Vacca, *Network and System Security (Second Edition)* (pp. 57-82). Syngress.