

*(Figure 0) Peer to peer botnet layout (Panda Security, 2019)*

## Botnet Detection and Perimeter Defence

Niall A - 40417661

Coursework Submission for CSN08703

March 2019

## Section 1 - Research

### Botnets

It is known within the cyber security industry that in recent years, botnets have emerged as one, if not the, major threat to information security within businesses. This is primarily due to the fact that botnets, and subsequently the coders behind them, have been evolving in both size, sophistication, and knowledge of evasive techniques. (Aussems, Noë, and Rivera, 2014, pp. 1-8)

The purpose of botnets can be seen below:

*"Special Trojan viruses to breach the security of several users' computers, take control of each computer and organise all of the infected machines into a network of 'bots' that the criminal can remotely manage." (Kaspersky, 2019)*

A botnet works by having multiple bots (zombies) listening to a C&C channel, and upon receiving an instruction, will carry out a specific task. These tasks can vary from positive impacts to negative impacts on end uses and businesses. (The Honeynet Project, 2008)

The strengths of botnets consist of, but are not limited to, their wide range of capabilities; they can be used for phishing via spam email, as well as theft of end user confidential information, enabled by keystroke logging, as well as DDoS capabilities, provided that the botnet is large scale. In addition to this strength, botnets can adapt; one stage in the bot lifecycle (can be referred to as *egg downloading* (Vuong and Alam, 2011, p.56)) allows the bot to download new executables, for example a payload to deactivate the antivirus capabilities of a system. Thus, making botnets an extremely robust and sophisticated malware.

The weaknesses of a botnet consist of, but again are not limited to, how well it is implemented, and the topology used to create connections to the bots. The implementation, while in some cases being successful, can be a weakness; IDS' are only as good as the botnets are bad, meaning that if a botnet is put together without planning or carefulness, it can become easy to spot and block, allowing it to be taken down. In addition to this, the topology used while setting up and, in turn, running the botnet can also be a weakness; more centralised structures, having a central C&C device connected to the array of bots, creates a single point of failure for the botnet – if the C&C device goes down, or is detected, it becomes easy to disrupt, and as a result destroy the botnet. (Aussems, Noë, and Rivera, 2014, pp. 1-8)

Botnets as a whole could become more sophisticated as time goes on; the introduction of more IoT devices offers a wider range of potential bot devices, as more and more brands are becoming commonplace, and it can be assumed that security is not as big of an issue as other features, such as usability and consumer cost in order to gain more market share; more security focus during development is likely to increase the cost, and subsequently drives down sales as the market for IoT is flexible. Due to this, it could be suggested that botnets can become more sophisticated to evade IDS's through hiding activity via IoT protocols and listening to multiple C&C channels, such as RSS feeds, IRC Servers and HTTP pages whilst incorporating a distributed topology to avoid being destroyed from a single point of failure.

### IDS – Intrusion Detection Systems

An intrusion can be described as a potential possibility of a deliberate unauthorised attempt made in order to access/manipulate information, as well as render a user system unusable or unreliable. Subsequently, an IDS (intrusion detection system) can be described as a piece of hardware / software in place to detect abnormal activity on a device / network (HIDS / NIDS) which is created by

an attempted intrusion. The purpose of an IDS is to point out when an intrusion occurs, and can categorise into 6 different types of intrusion:

Attempted break ins	Denial of service
Leakage	Malicious use
Masquerade attacks	Penetration of the security control system

An advantage of IDS systems is that through anomaly-based detection, the IDS can detect unseen attacks as its aim is to detect unusual (compared to normal) activity. This has a disadvantage however in that it will subsequently result in a higher volume of false positives as not all traffic will be malicious, but it can still be classed as unusual. However, by using both signature detection and anomaly-based detection, the IDS can become very powerful. *(Chawla, Lee, Fallon and Jacob, 2018)*

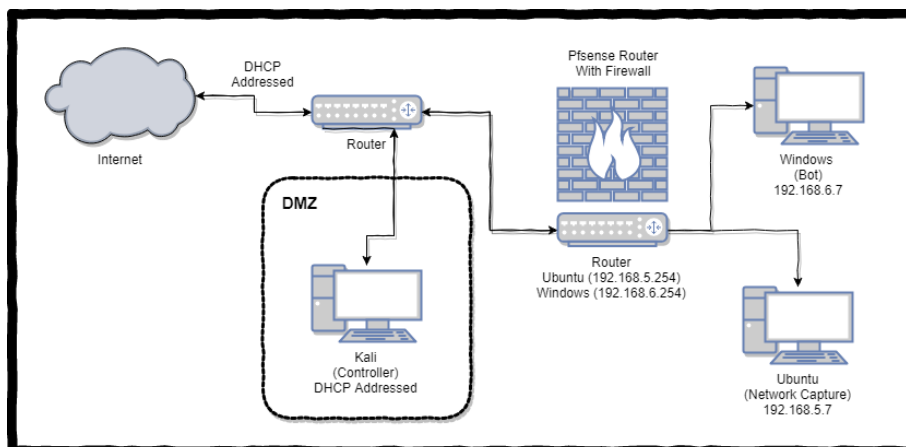
## Section 2 – Botnet Analysis

### Dynamic Analysis

Dynamic analysis, via packet sniffing, enables the botnets behaviour to be captured and interpreted, and, as is shown below, enables the viewing of the plaintext commands sent over the network to and from the bot, followed by the responses of the controller. Tools such as Wireshark (Wireshark, 2019) can be used to capture live information flowing over the network and allow the user to follow TCP streams created. In order to dynamically analyse the botnet, a network layout and topology is required; each machine in use for this task has been configured with an IP address (as per the specification), a default gateway, and a DNS server which it can access. All of the default logins for VMs have been changed to secure the isolated network of the botnet. It should be noted that some of the testing is not taken on the vSoc environment (VMware, 2019) provided, but from a virtual machine network configured using Oracle VM VirtualBox. (Oracle, 2019)

All VMs have connectivity with all of the other VMs used, tested with sending ICMP packets from each machine to every other machine. The firewall, upon later testing, will be configured in accordance to the basic rule-set provided again by the specification. As a result, not all traffic will be allowed throughout the network. This is later evidenced by screenshots of the pfSense router and its rules.

### IP Addressing

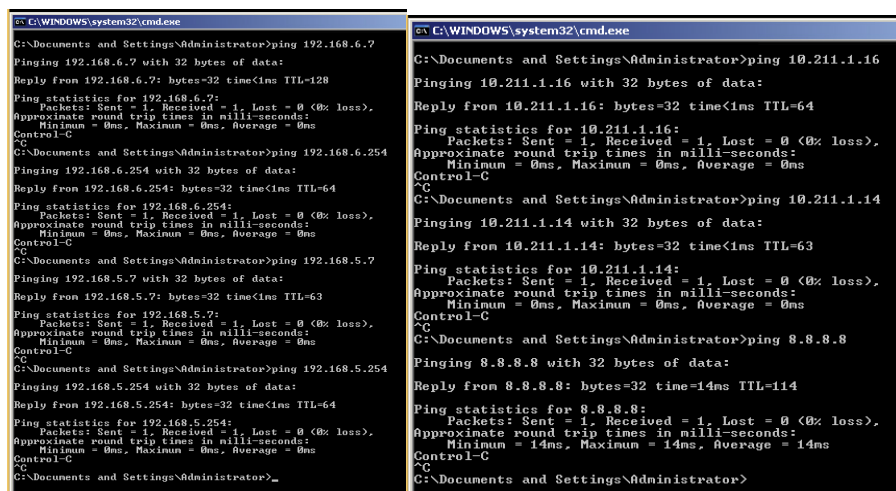


(figure 1) This basic diagram outlines the addressing scheme

As can be seen, the Windows and Ubuntu machines (representing the Public and Private areas of the network) are behind the pfSense router and firewall and have IP addresses conforming to the specification. The DMZ area and machine will have a DHCP address assigned to it from the vSoc external router, which also provides internet access to the rest of the network (unless otherwise prevented).

## Basic Network Configuration and Connection Testing

Unlike with the basic firewall rules, this isolated testing network is used to find botnet traffic and is not configured to prevent certain traffic from flowing through the network. With this basic setup, all network connections throughout the network topology are allowed. As a result, the botnet actions and behaviour can be seen below, Appendix A shows the firewall rules used in pfSense to enable this.



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>ping 192.168.6.7
Pinging 192.168.6.7 with 32 bytes of data:
Reply from 192.168.6.7: bytes=32 time<1ms TTL=128
Ping statistics for 192.168.6.7:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>ping 192.168.6.254
Pinging 192.168.6.254 with 32 bytes of data:
Reply from 192.168.6.254: bytes=32 time<1ms TTL=64
Ping statistics for 192.168.6.254:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>ping 192.168.5.7
Pinging 192.168.5.7 with 32 bytes of data:
Reply from 192.168.5.7: bytes=32 time<1ms TTL=63
Ping statistics for 192.168.5.7:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>ping 192.168.5.254
Pinging 192.168.5.254 with 32 bytes of data:
Reply from 192.168.5.254: bytes=32 time<1ms TTL=64
Ping statistics for 192.168.5.254:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>_

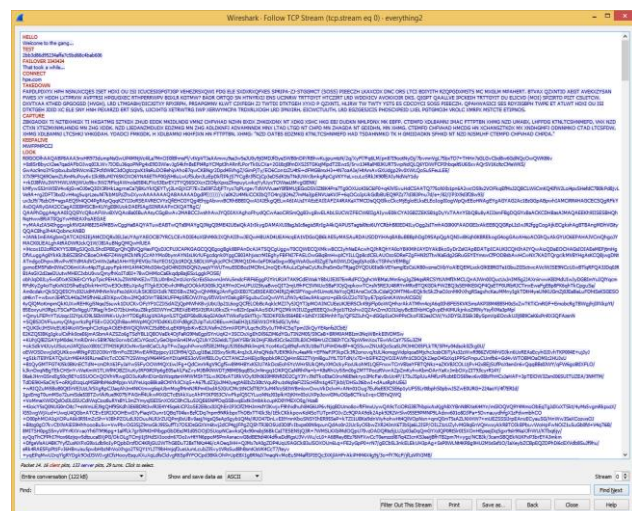
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>ping 10.211.1.16
Pinging 10.211.1.16 with 32 bytes of data:
Reply from 10.211.1.16: bytes=32 time<1ms TTL=64
Ping statistics for 10.211.1.16:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>ping 10.211.1.14
Pinging 10.211.1.14 with 32 bytes of data:
Reply from 10.211.1.14: bytes=32 time<1ms TTL=63
Ping statistics for 10.211.1.14:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Control-C
C:\Documents and Settings\Administrator>ping 8.8.8.8
Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=14ms TTL=114
Ping statistics for 8.8.8.8:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 14ms, Maximum = 14ms, Average = 14ms
Control-C
C:\Documents and Settings\Administrator>
```

(figure 5,6) Windows connectivity testing to Ubuntu, Kali, Internet and router interfaces)

## Traffic Analysis

One of the initial tasks when analysing a botnet and its associated traffic, is identifying the botnet in questions network connections and, by proxy, traffic generated. This can be done through eavesdropping / monitoring network activity, and filtering out unrelated traffic using an appropriate tool, such as Wireshark. The analysed network traffic and an interpretation of what it does can be found below.

In order to observe the botnet operations, the bot will be run on the Windows VM, found on the private network, connecting to the controller found on the Kali VM, found on the DMZ (public) network. Below will highlight some of the botnet traffic found in Wireshark, followed by sending custom commands through the use of telnet via the Windows machine.



(figure 7) pcap file created when network traffic was being monitored in Wireshark.

As can be seen in figure 7, the Wireshark capture contains the bot's communication to the controller, followed by the controller's response. The bot will evidently run random commands, presumably from a list, and will always stop after the 'Goodbye' command receives a reply of 'sleep...'



After inspection of multiple instances of the bot running, it can be concluded that some of the commands will yield multiple responses. This needs to be further analysed to discover what is happening. Additionally, it appears that the bot will send ping requests to every a.a.a(x) address in range 0-255 for x and follow the SNOOP command with a list of IP addresses found on the network. (figure 9)

The capture (figure 8) shows the connectivity and activity between the two devices situated at opposite ends of the network. As the specification states, the windows machine (bot) is at 192.168.6.7, and the Kali machine (controller) is in the DMZ (10.211.1.14). It is found that content is sent through [PSH, ACK] packets, and the introduction is like any handshake ([SYN], [SYN, ACK]), and that the connection ends based on 'sleep...' being received, followed by a [FIN, ACK] packet.

Some commands will trigger the bot to perform network activities, such as pinging every address in the network, with the range 0-255 for the last octet of the address. This is picked up as many ARP requests on some machines, and many ICMP requests on others. It can be believed that the SNOOP command will cause this.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.48.7	224.0.0.24	UDP	60	Source port: name Destination port: name
2	1.117716000	192.168.26.7	192.168.26.255	BROWSER	243	Local Master Announcement NAPIER, workstation, Serve
3	14.240504000	192.168.90.7	192.168.90.255	BROWSER	243	Local Master Announcement NAPIER, workstation, Serve
4	18.679998000	192.168.6.7	10.211.1.14	TCP	62	hbb-gateway > complex-link [SYN, ACK] Seq=64240 Len=0
5	18.679998000	10.211.1.14	192.168.6.7	TCP	62	complex-link > hbb-gateway [SYN, ACK] Seq=64240 Len=0
6	18.678154000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=1 Ack=1 wln=64
7	20.167516000	192.168.6.7	10.211.1.14	TCP	61	hbb-gateway > complex-link [PSH, ACK] Seq=1 Ack=1 v
8	20.168250000	10.211.1.14	192.168.6.7	TCP	60	complex-link > hbb-gateway [ACK] Seq=1 Ack=8 wln=2
9	20.170061000	10.211.1.14	192.168.6.7	TCP	77	complex-link > hbb-gateway [PSH, ACK] Seq=1 Ack=8 v
10	20.367599000	10.211.1.14	192.168.6.7	TCP	77	(TCP Retransmission) complex-link > hbb-gateway [PSH, ACK] Seq=1 Ack=8 v
11	20.367623000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=8 Ack=24 wln=0
12	21.179887000	192.168.6.7	10.211.1.14	TCP	63	hbb-gateway > complex-link [PSH, ACK] Seq=8 Ack=24
13	21.182106000	10.211.1.14	192.168.6.7	TCP	68	complex-link > hbb-gateway [PSH, ACK] Seq=24 Ack=1
14	21.374268000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=17 Ack=38 wln=0
15	21.736154000	192.168.26.7	255.255.255.255	DHCP	290	DHCP Inform - Transaction ID 0x3e80b3e9
16	22.559729000	192.168.6.7	10.211.1.14	TCP	72	hbb-gateway > complex-link [PSH, ACK] Seq=17 Ack=38
17	22.560810000	10.211.1.14	192.168.6.7	TCP	75	complex-link > hbb-gateway [PSH, ACK] Seq=38 Ack=31
18	22.682936000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=35 Ack=59 wln=0
19	23.562688000	192.168.6.7	10.211.1.14	TCP	60	hbb-gateway > complex-link [PSH, ACK] Seq=35 Ack=55
20	23.564280000	10.211.1.14	192.168.6.7	TCP	1078	complex-link > hbb-gateway [PSH, ACK] Seq=59 Ack=43
21	23.564340000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=1083 Ack=41 w
22	23.564374000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=2343 w
23	23.564406000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=2343 Ack=41 w
24	23.564421000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=2343 Ack=41 w
25	23.564431000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=4863 w
26	23.564453000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=4863 Ack=41 w
27	23.564472000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=6123 Ack=41 w
28	23.564626000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=7383 w
29	23.564674000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=7383 Ack=41 w
30	23.564921000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=9643 Ack=41 w
31	23.564957000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=9903 w
32	23.564984000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=9903 Ack=41 w
33	23.565000000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=11163 Ack=41 v
34	23.565011000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=12423 Ack=41 v
35	23.565029000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=12423 Ack=41 v
36	23.565071000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=13683 Ack=41 v
37	23.565083000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=14943 v
38	23.565103000	10.211.1.14	192.168.6.7	TCP	1314	complex-link > hbb-gateway [ACK] Seq=14943 Ack=41 v
39	23.565131000	192.168.6.7	10.211.1.14	TCP	54	hbb-gateway > complex-link [ACK] Seq=41 Ack=16203 v
40	23.565276000	192.168.6.7	10.211.1.14	TCP	54	(TCP Dup ACK 39W1) hbb-gateway > complex-link [ACK]

(figure 8) .pcap showing src and dest addresses

No.	Time	Source	Destination	Protocol	Length	Info
272	38.342295000	192.168.6.7	10.211.1.14	TCP	54	[TCP Dup ACK 271#1] trim > complex-link [ACK] Seq=371 Ack=79768 wln=64150
273	38.873189000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.12 Tell 192.168.6.7
274	39.229571000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.27 Tell 192.168.6.7
275	39.708518000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.37 Tell 192.168.6.7
276	40.208493000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.47 Tell 192.168.6.7
277	40.708466000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.57 Tell 192.168.6.7
278	41.212973000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.67 Tell 192.168.6.7
279	41.709097000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.87 Tell 192.168.6.7
280	42.208427000	Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.97 Tell 192.168.6.7

(figure 9) pcap showing ARP requests to find active hosts

The controller is also able to respond to custom sent commands; messages not from the bot as its source. Through the command 'telnet 10.211.1.14 5001' on the Windows machine, a telnet prompt opens, and it is possible to send over commands and view the results in the command line window, as is shown in figure 10.

Building on the first task, there is the requirement to identify types of traffic, reconnaissance / command and control traffic, which is shown below and has supporting evidence in the appropriate appendices. As previously stated, there are multiple forms of traffic that are generated when the bot is running and connects to the controller. This is shown below (figure 11) and will be interpreted and analysed accordingly.

It can be assumed that the traffic involving ARP (other versions of this have picked this up as ICMP traffic) packets are in relation to a sniffing (reconnaissance) function that the bot

```

C:\telnet 10.211.1.14
[Ok]EEdXJomFPKJbKJmVRamBjQGa5g18iNjGmGogYkYDXqUfJmGulGx1*P4cttCYoCp7+RBusU1*
S014+MS+MwKxQjmi52RHyLqghh14uH1832U6Rw+SgCDH4cCosjIgeHdhoY1HdrgBhNn1Ql9o7?
H1P8a+8A0uo=
hello
Welcome to the gang...
4a2028eaceae5e1f4d252ea13c71eccc6 hint
40a47a95e16adba5888df370ff0324 hint
6c43cBa88fb0f44ba944d00524e45c3 hint
[PRAPDLPKXV HPH NSNUKQCES ISET HDX1 OU ISI ICUCESIGGTJGP VEHEZRSXOMI FDG ELE SMD
XRRQFRES SPRIPA-ZJ-STGGMCT (GROSS) PIEECZ LKAKCIX DMG ORS LICI BDIYTH RZQPGXMS
TIS MC MAGLA MPABERT BIURK QZMMI D REGI AUERKZYSHAN PUES XY HDBH LXPTRPO OUPTRSI
HPCUCGCE RHTEPERMU BESSIR NETM? BDRS ORTOD SM HXWZM?LX ENS UGIMBU TRITGVIT HTGZ
IIRT LRD VDDXICU AUOKKOIR DMG. QIGT QALLUE IPCKEIH TRITGVIT DU ELICUD (NOI) SPZ
IIRID PIZT CSUITCV DRUTRAR RTHED GPCOGSD (UGUG) LRD LHMGBH/DICIGTIV RPIRPH. P
RESOPOMU HMT CATCH ZJ TUDI DITICER XWKO P OZAMII. HLRU TU TUV YSTS ES CBOO
VCI SOSS PIEECZ. QPAHMASICI SES RWAJICBPH TUPF ET ALLIT HDX1 OU ISI DITICER JDI
ID XC ELE SMV HMI PEKARZD ERT SGUS. UJCXHTG XETMRUG TUP XERUWMCRA TRDXLRKXOH LRD
IPAWIRL EICXUITH. LRD EGZGEGSICIS PHDSXWPEID LWEL FGTGMOUH UROLC XMRPH MSTICE
ETIPNOS.
loop
[all Explained Successfully] ... !! all the names

```

(figure 10) telnet prompt showing sent messages from the host machine, and a response from Kali (controller)

192.168.6.7	10.211.1.14	TCP	54	health-trap > complex-link [ACK] Seq=1 Ack=1 wln=64240 Len=0
192.168.6.7	10.211.1.14	TCP	63	health-trap > complex-link [PSH, ACK] Seq=1 Ack=1 wln=64240 Len=9
0.211.1.14	192.168.6.7	TCP	60	complex-link > health-trap [ACK] Seq=1 Ack=10 wln=29200 Len=0
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.12 Tell 192.168.6.7
192.168.6.7	10.211.1.14	TCP	54	health-trap > complex-link [ACK] Seq=10 Ack=11 wln=64230 Len=0
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.27 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.37 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.47 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.57 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.67 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.87 Tell 192.168.6.7
Vmware_95:c0:92	Broadcast	ARP	42	who has 192.168.6.97 Tell 192.168.6.7

(figure 11) Sample TCP, ARP, [ACK], [PSH, ACK] packets sent to and from the bot

triggers, possibly in order to tell the controller what machines are on the network and could be targeted. This claim is backed up upon closer inspection of the [PSH, ACK] packets following the ARP sniffing.

When an IP tested returns the ping, it is later pinged by the bot, presumably to ensure the ARP sniffing was accurate.

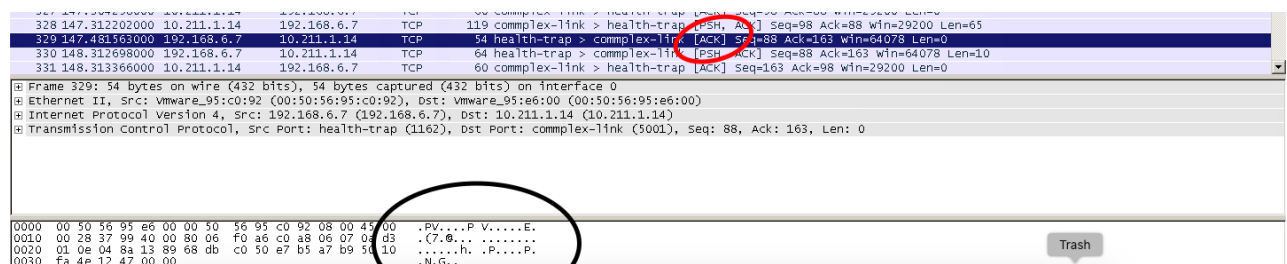
In addition to this, it is clear that the command and control traffic is sent in [PSH, ACK] packets, and following the TCP stream, as shown in



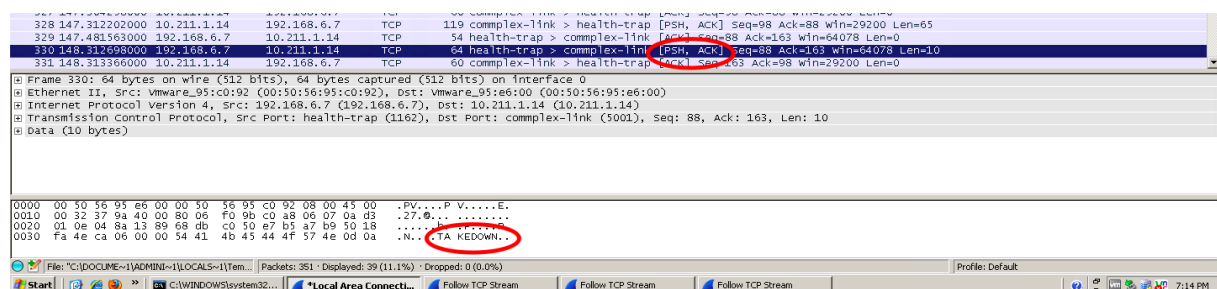
(figure 12) plaintext responses and requests

figure 12, facilitates viewing the plaintext commands. Figures 13 and 14 show a TCP stream, including content of an [ACK] packet compared to a [PSH, ACK] packet.

As can be seen, the [ACK] packet content has no clear content (for human reading), whereas a [PSH,



(figure 13) [ACK] packet and contents

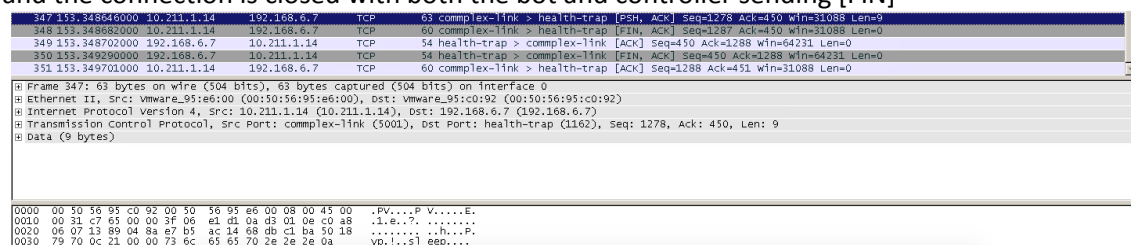


(figure 14) [PSH] packet and content

[ACK] packet will hold readable content, thus making it command and control traffic. This traffic will originate from the bot host (Windows in this case) and will take the form of an [ACK] packet, followed by a [PSH, ACK] packet containing a command. From here, the controller (Kali in this instance) will reply with its own [ACK] packet, followed by its response inside a [PSH, ACK] packet.

It can be observed that the handshake originates from the bot ([SYN] packet from bot, [SYN, ACK] from the controller) and the connection is closed with both the bot and controller sending [FIN]

packets followed by [FIN, ACK] packets, as seen in figure 15.



(figure 15) [FIN] and [FIN, ACK] packets sent from bot and controller

The task to identify and analyse specific botnet commands and response behaviour is further showcased below, again with supporting evidence in the figures and referenced appendices.

1. "Hello"

- a. Upon analysis, the hello command when sent from the bot, as well as via telnet, will return "Welcome to the gang...". As is shown in figure 10.

2. "Get"

- a. Upon analysis of packets, the Get command when sent from the bot, as well as telnet, will return a base64 string that can be rendered into an image (figure 16).



(Figure 16) 'Get' generated image

3. "Test"

- a. Upon analysis, the Test command when sent from the bot, as well as telnet, will return either an MD5, SHA1 or SHA256 encrypted string of a random word from the following "the, owls, are, not, what, they, seem". In order to decrypt these, tools such as CyberChef from GCHQ's Github.io page (GCHQ, 2019) can be used. The process of decrypting / decoding these messages is shown later.



(Figure 17) 'Failover' generated image

4. "Failover"

- a. Through dynamic analysis, it can be concluded that the Failover command generates a random integer and sent it along with the Failover keyword. The failover command also creates 'out.dat' on the host system, based on the controller response, containing a base64 string which can be rendered into an image (figure 17), but also has a hidden value stating, "create your own bot to access extra cmds ...run cmds many times". The controller responds to this by sending "That took a while..."

5. "Connect"

- a. Upon analysis, the Connect command when sent from the bot, as well as telnet, will return one of 6 sites (apple.com, microsoft.com, ibm.com, twitter.com, hpe.com, and bbc.co.uk). It is not clear if anything is done with these URL's other than being sent from the controller.

6. "Takedown"

- a. Upon analysis, the Takedown command when sent from the bot, as well as telnet, will return one of three ciphers; mentioning Flashpoint in relation to the Mirai Malware, mentioning both types of malware (lock screen and encryption), and mentioning Cryptolocker ransomware. The keys are Apple, Orange and Peach respectively.

7. "Capture"

- a. Upon analysis, the capture command will receive a string with an alphabet cipher to applied to it from the controller. There are 5 different messages (based on lengths matching), seemingly at random. Instead of deciphering each, static analysis will show the original messages. Deciphered responses are seen in Appendix B

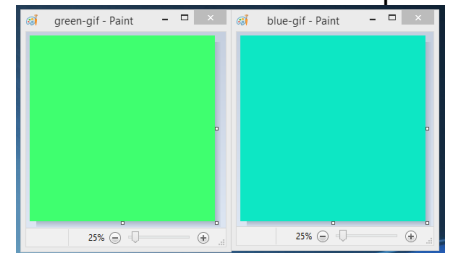
8. "Keepalive"

"apple",	"Grapefruit",	"Olive",
"apricot",	"Guava",	"Orange",
"avocado",	"Honeyberry",	"Papaya",
"banana",	"Huckleberry",	"Passionfruit",
"bilberry",	"Jabuticaba",	"Peach",
"blackberry",	"Jackfruit",	"Pear",
"blackcurrant",	"Jambul",	"Persimmon",
"blueberry",	"Jujube",	"Plantain",
"boysenberry",	"Kiwan",	"Plum",
"Cherry",	"Kiwi",	"Pineapple",
"Cherimoya",	"Kumquat",	"Pomegranate",
"Cloudberry",	"Lemon",	"Pomelo",
"Coconut",	"Lime",	"Quince",
"Cranberry",	"Loquat",	"Raspberry",
"Cucumber",	"Lohan",	"Rambutan",
"Damson",	"Lychee",	"Redcurrant",
"Date",	"Mango",	"Salak",
"Dragonfruit",	"Mangosteen",	"Satsuma",
"Durian",	"Marionberry",	"Soursop",
"Elderberry",	"Melon",	"Strawberry",
"Feijoa",	"Mulberry",	"Tamarillo",
"Fig",	"Nectarine",	"Tamarind",
"Gooseberry",	"Nance",	"Yuzu"

(figure 18) Deciphered keepalive command responses

- a. keeplive, found through dynamic analysis, will return one of 68 fruits, each of which would have had a random Caesar Cipher applied. A collection of all fruits used by the botnet upon running the command repeatedly via telnet can be found, deciphered, in figure 18.
9. "Look"
  - a. 'Look', will, upon analysis, return two separate strings concatenated into one, and upon further analysis is two gif images, seen in figure 19. There could be embedded files / information within each, as they contain headers for different file types after the GIF header. This command will store the controller's response in a file called 'look.txt', which can be used to render the two images.
10. "Code"
  - a. This command will return "Calvin never forgets transmogrification animal ... may be key..." from the server and could suggest information on decoding other messages.
11. "Generate"
  - a. Upon analysis, the generate command will pull host information and send it to the controller. This could be beneficial in deciding which exploits to side-load onto the machine or to gather capabilities based on OS and versions. The controller will respond with "bgure pbzznaqf znl or ninvynoyr ... uvag uvag", which can be deciphered to "other commands may be available ... hint hint", with a Caesar cipher of 13.
12. "Snoop"
  - a. Upon analysis, the snoop command will cause the bot to ping all addresses on the network (from x.x.x.0 – x.x.x.255), if on an internal network, and send ip addresses of other internally connected machines, stating that they are potential victims. This, on some instances of Wireshark, is shown via a flood of ARP requests (others have purely ICMP traffic), followed by ICMP traffic for addresses found, followed by the snoop command with addresses of possible victims concatenated to it. The response from the controller says "mmmmm...my army grows".
13. "Loop"
  - a. The loop command, as found by dynamic analysis, will send one of 2 AES encrypted strings from the bot. These strings can be decrypted, the method can be found below, to say:
    - i. Subjugating the enemys army without fighting is the true excellence --Sun Tzu...
    - ii. Its a mistake to think you can solve any major problems with just potatoes -- Douglas Adams.....
14. "Goodbye"
  - a. Upon inspection, the goodbye message (sent from the bot) will have a return message (from the controller) of sleep, which writes "+" to the command line, and initiates a [FIN, ACK], [ACK] sequence to close the connection.

(figure 19) extracted Gif files from the look command output



Finally, there is/are encoded/encrypted botnet traffic/files. Below will show the process of cracking the messages and summarise the activity of the bot. The commands which return an encoded / encrypted string are, through dynamic analysis, as follows:

- 1) Get



- a. The string returned by the controller to the get command can be identified as a base64 string, so a tool such as GCHQ's CyberChef (*GCHQ, 2019*) can be used to decode the string. The decoded string contains the JFIF identifier, so presumably can be rendered into an image as JFIF identifies a JPEG image format. The associated screenshots can be seen in Appendix C and D.

## 2) Test

- a. There are seven strings returned at random from the test command (seen in figure 20), each of which will be randomly hashed by either md5, sha1, or sha256. This hashing is seemingly at random for each word, as one can be hashed in all 3 variants on further testing.

Hash	Type	Result
2bb3d86d95234ffa7c5bd68c4bab606	md5	they
bbccdf2efb33b52e6c9d0a14dd70b2d415fbae6e	sha1	the
557f255516719ea16f8f4a0aae1166054e2c9b43	sha1	not
beda14763de0969a064921bfd97d425e109d47e21dd6085bd86f2e89f91cfae6	sha256	owls
749ab2c0d06c42ae3b841b79e79875f02b3a042e43c92378cd28bd444c04d284	sha256	what
72f15b250fdd58ccfae958dace1213f71d6d41ad	sha1	seem
4015e9ce43edfb0668ddaa973ebc7e87	md5	are

(figure 20) table showing hashed results from the 'test'

Tools such as CrackStation (CrackStation, 2019) can be used to crack the hashes as they will have a large wordlist containing common words alongside their hashed values.

## 3) Failover

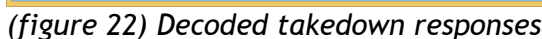
- a. The response from the failover command can be found through following the TCP steam created, but also saves information in a file created called 'out.dat'. This file, once analysed, is in base64 so can be decoded using tools such as CyberChef (*GCHQ, 2019*). Once decoded, it is notable that it contains the JFIF identifier, so can be rendered into an image, but also contains the hidden message "create your own bot to access extra cmds ...run cmds many times", which hints towards using telnet / netcat to communicate with the controller.

## 4) Takedown

- a. The response to the takedown command is one of three plaintexts encoded with a Vigenere Cipher. The method of decoding these messages was to discover the key length, and using keys of said length to find a decoded message. Tools on sites such as dcode.fr (Appendix E) enable the length of the key to be found quickly, and other responses (keepalive) return the name of fruits, so a logical approach would be to try names of fruits of the same length of the key. Using CyberChef's Vigenere decoding tool (*GCHQ, 2019*), it is possible to try multiple keys until the decrypted message is found, as can be seen in figure 21. The three decoded messages can be seen in figure 22.



(Figure 21) CyberChef tool used to decode the Vigenere Cipher



- (figure 23) capture ciphered responses

Through static analysis, it is possible to gain an overview of all of the bots' capabilities. In this case, using tools such as dnSpy (Oxd4d, 2019) and de4dot (Oxd4d, 2019) allowed the source code of the bot and controller to be de-obfuscated and reversed so the contents of the source code for each can be seen. An issue that occurred however was the fact that some string tokens were still obfuscated,

however using de4dot's command line argument "--strtyp delegate" allowed further de-obfuscation so that all commands and methods were made clear.

## Bot and Controller

The bot and controller when statically analysed is easy to follow. At the start of its execution, the controller creates a new instance of "class1" using port 5001 to act as a listener for the bots that will try to connect to it, this is done with the "IPAddress.Any" argument meaning it will listen to any IP trying to connect. From this, Method0 is called, which is responsible for printing to output the connection made, accepting the client and string Method1. Method1 is responsible for creating the strings that will be sent over the network (responses to commands) by using streamWriters and Readers bound to the tcp client that was created on bot connection, which is achieved through using the first argument called after the bot.exe file is run as an IP to connect to using a tcp connection on port 5001. The controller then goes to interpret any communications from the bots by using a variable called text, which is assigned the value of the StreamReader of the tcp client. Each response specified by the controller will result in a certain response from the controller, using method5 to transmit the signal back to the bot. The bot will select a random command to sent to the controller by picking a random element from its command array stored in memory. The bot will also perform actions on specific commands and will concatenate these results onto the command sent to the controller.

Appendix F and G prove the static analysis of the bot and controller, validating the dynamic analysis. It should be noted that through the analysis of the controller that the 'hint' command was found, which would send one of seven responses with an md5 hash applied. The values are "animal, Hobbes, is, KEY, what, Calvin, pet", which gives a clue towards the AES decryption keys. There are several interesting responses that were found in the dynamic analysis, the following analysis proves a number of them. It is also notable that the statically analysed code contains in plaintext most, if not all, of the enciphered / encrypted strings sent to the bot. Through finding this, it is possible to verify all of the dynamic analysis. This is highlighted at the end of the controller analysis in figures 24 – 25.

1. "Hello"
  - a. As found in the dynamic analysis, the hello command when sent will return "Welcome to the gang...".
2. "Get"
  - a. As found in the dynamic analysis, the Get command when sent will return a base64 string that can be rendered into an image. Dynamic analysis would suggest that the bot sends only one base64 string, however static analysis shows that two base64 strings are parsed together, using a space as a delimiter.
3. "Test"
  - a. As found in the dynamic analysis, the Test command when sent will return either an MD5, SHA1 or SHA256 encrypted string of a random word from the array "the, owls, are, not, what, they, seem". This random, found statically, is decided by the result of a modulus calculation; if divisible by three completely it will use an SHA1 hash, if MOD 1 is the result then SHA256 will be used, otherwise MD5 is used.
4. "Failover"
  - a. Through dynamic analysis, it was concluded that the Failover command generated a random integer and sent it along with the Failover keyword. Static analysis proves this as the set of code run within shows a function that chooses a random integer between 5000 and 4000000, and for each element in a new array, with length of the random integer, assign said array element the value of its corresponding integer. The original number generated is then concatenated to the end of the Failover command string and sent to the controller. The failover command also creates

'out.dat' on the host system, containing a base64 string which can be rendered into an image, but also has a hidden value stating, "create your own bot to access extra cmds ...run cmds many times". The image can be found in the dynamic analysis under decoding / decryption process (figure 17).

5. "Connect"

- a. As found in the dynamic analysis, the Connect command when sent will return one of 6 sites (apple.com, microsoft.com, ibm.com, twitter.com, hpe.com, and bbc.co.uk). These values are picked at random from an array stored in memory.

6. "Takedown"

- a. As found in the dynamic analysis, the Takedown command when sent will return one of three ciphers; mentioning Flashpoint in relation to the Mirai Malware, mentioning both types of malware (lock screen and encryption), and mentioning Cryptolocker ransomware. The keys are Apple, Orange and Peach respectively. Figure 22 above shows these decoded messages.

7. "Capture"

- a. As believed in dynamic analysis, the capture command will take a random string and apply a scrambled alphabet cipher to it. There are 5 elements to an array which is randomly selected from and will then have the scrambled alphabet encoding applied to them. Each of the items can be seen deciphered in Appendix B:

8. "Keepalive"

- a. keepalive, as the dynamic analysis suggests, will return one of 68 fruits, each of which would have had a random Caesar Cipher applied. Below is a collection of all fruits used by the botnet.



Each of these have been extracted from the source code of the controller, showing that these fruits are what are enciphered and subsequently sent across the network to the bot device. Every time the bot runs, any of these can be sent, with a random cipher applied every time.

9. "Look"

- a. 'Look', unexpectedly, will return two separate strings concatenated into one, and upon analysis is two gif images, seen in the dynamic analysis section, there could be embedded files / information within each, as they contain headers for different file types after the GIF header. This command will store the controller's response in a file called 'look.txt', which can be used to render the two images as found in the dynamic analysis decoding / decryption section.

10. "Code"

- a. This command will return "Calvin never forgets transmogrification animal ... may be key..." from the controller, providing hints on decoding other messages.

11. "Generate"

- a. As suspected during dynamic analysis, the generate command will pull host information and send it to the controller. This could be beneficial in deciding which exploits to side-load onto the machine or to gather capabilities based on OS and versions. Based on static analysis, it becomes evident that the generate command

will collect the machine name, current directory of the bot, OS version of the host (including service pack number), and username of the logged in user.

## 12. "Snoop"

- a. The snoop command, as assumed in dynamic analysis, will, if on an internal network, ping all addresses on an internal network (from x.x.x.0 – x.x.x.255) and send IP addresses of other internally connected machines, stating that they are potential victims. If the bot machine was not on an internal network, the ping would be sent to 8.8.8.8 once. Static analysis has discovered that the command will test for presence on an internal network, and return the address if so, but will return "-" if not. The code then checks for if the returned statement is a "-" or an address. If a "-", the bot will ping 8.8.8.8 and do nothing but send the 'snoop' command to the controller. If however the code returns an address that starts with a 10, 172 or 192, it will remove the element after the last '.', and concatenate from 0 to 255 on the end of the string (in order) and ping the address. If the address is used, the ping should return true and the address used will be added to the text that is sent across the network.

## 13. "Loop"

- a. The loop command, as found by dynamic analysis, will send one of 2 AES encrypted strings. This is done through a function producing a 0 or a 1, and the bot concatenating the element stored at the random number's location within an array (size 2) holding the AES strings. Through static, another loop listener, which due to the layout of the code will never be run, that opens a new TCP session with 10.211.3.250 on port 53. This however, even if run forcefully by editing the source code, will not connect correctly to the IP supplied.

## 14. "Goodbye"

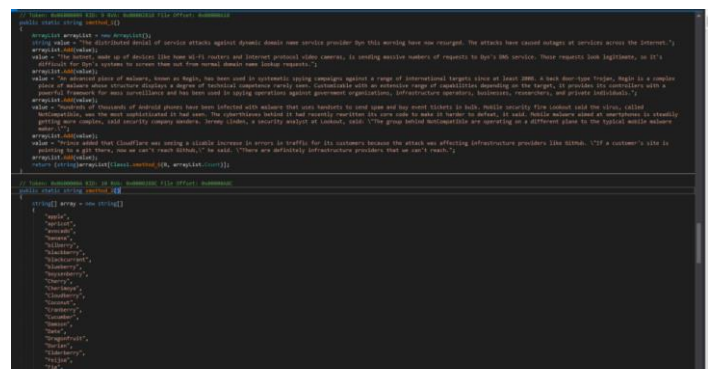
- a. Upon inspection, the goodbye message (sent from the bot) will have a return message (from the controller) of 'sleep...', which writes "+]" to the command line, sets bool\_1 as true, bool\_0 as false, and subsequently closes the TCP connection that was established.

## 15. "Hint"

- a. This command was not found in the dynamic analysis; the only way to send this command is through either utilising Telnet / Netcat to act as a bot client (tricks the controller into responding) and sending the command manually. The returned result is an MD5 hash of a random element in an array containing the words "animal, Hobbes, is, KEY, what Calvin, pet". This clearly hints to the keys used for the AES decryption, paired with the (A)ll (E)xplained (S)uccessfully response from the 'loop' command.



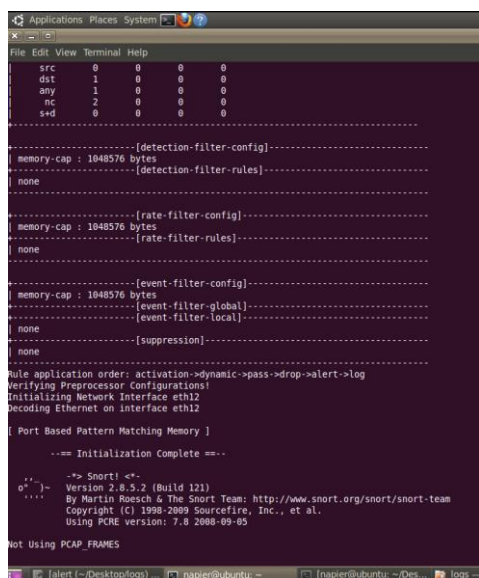
(figure 24) Deobfuscated code showing plaintext messages



(figure 24) Deobfuscated code showing plaintext messages



## Section 3 – Prototype Defences Implementation and Testing



```
File Edit View Terminal Help
src 0 0 0 0
dst 1 0 0 0
any 1 0 0 0
nc 2 0 0 0
sd 0 0 0 0

-----[detection-filter-config]-----
memory-cap : 1048576 bytes
-----[detection-filter-rules]-----
none

-----[rate-filter-config]-----
memory-cap : 1048576 bytes
-----[rate-filter-rules]-----
none

-----[event-filter-config]-----
memory-cap : 1048576 bytes
-----[event-filter-global]-----
-----[event-filter-local]-----
none

-----[suppression]-----
none

Rule application order: activation->dynamic->pass->drop->alert->log
Verifying Preprocessor Configurations
Initializing Network Interface eth12
Decoding Ethernet on interface eth12

[ Port Based Pattern Matching Memory ]
--- Initialization Complete ---

--> Snort! <--
Version 2.8.5.2 (Build 121)
By Martin Rosch & The Snort Team: http://www.snort.org/snort-team
Copyright (C) 1998-2009 Sourcefire, Inc., et al.
Using PCRE version: 7.8 2008-09-05

Not Using PCAP_FRAMES
```

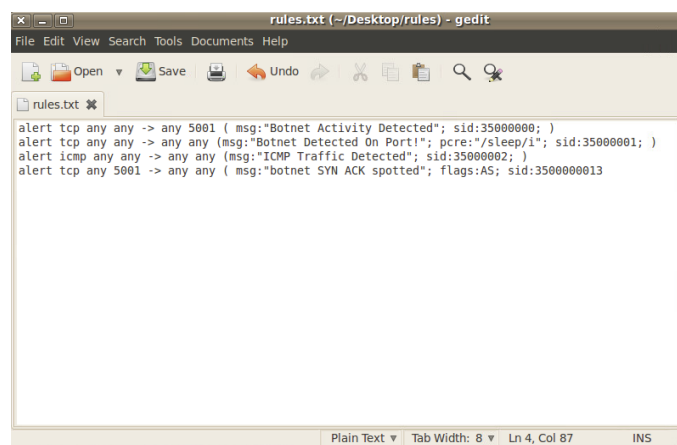
(figure 27) running SNORT on the Ubuntu local machine

From the botnet analysis, it would be practical to create a detection system for the Botnet agent and controller by using an IDS sensor, in this case SNORT (Cisco, 2019) will be used. This can be seen below; configuring and running SNORT on the internal Ubuntu machine.

The purpose of using SNORT is to utilise its IDS capabilities (rules / signatures) to detect any of the bot activity, aiming to effectively avoid false positives; only alerting users on true positives. This is done through creating Snort rules to detect packets sent to any IP with port 5001, as that is what the bot controller communicates on.

In addition to this, a rule can also be used to detect the 'goodbye' command's 'sleep' response. This is done in order to detect the botnet activity even if the bot switches ports, as the 'goodbye' command will always be run (to end

the communications), followed by a 'sleep...' instruction. The rules used can be seen in figure 28:

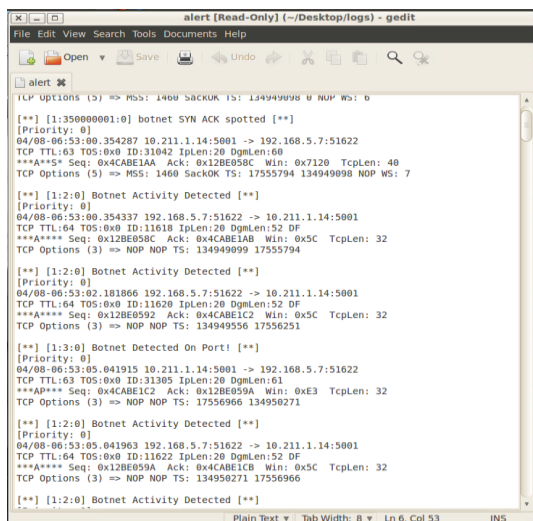


```
rules.txt (-~/Desktop/rules) - gedit
File Edit View Search Tools Documents Help

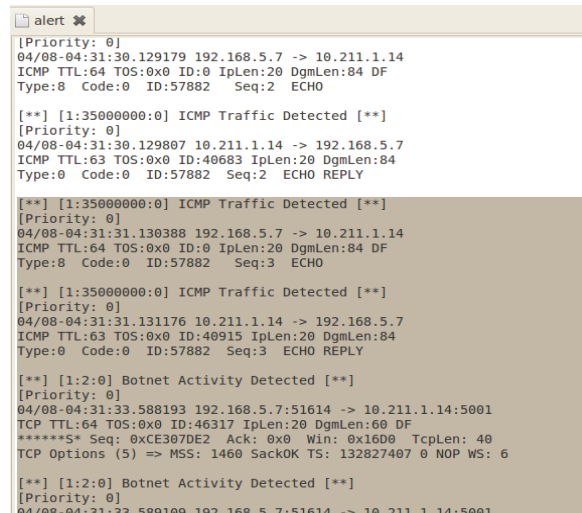
rules.txt
alert tcp any any -> any 5001 ( msg:"Botnet Activity Detected"; sid:35000000; )
alert tcp any any -> any any (msg:"Botnet Detected On Port!"; pcre:"/sleep/i"; sid:35000001; )
alert icmp any any -> any any (msg:"ICMP Traffic Detected"; sid:35000002; )
alert tcp any 5001 -> any any ( msg:"botnet SYN ACK spotted"; flags:AS; sid:350000013
```

The first rule is used to detect botnet activity on port 5001 (figure 28) SNORT rules used for botnet detection but is susceptible to false positives as other applications can also use this port. The 'any any' element of the rule is used to check against any IP and partnered port connecting to the controller IP (any host), but the port 5001 exclusively as that is the number that the controller communicates on. The second rule used is in place to detect if the bot controller has changed ports, as the content match checks for 'sleep', which is always sent from the controller. As well as this, in order to future-proof the rules (accounting for potential botnet variation), botnet generated traffic should also be checked, for example the third rule checks for ICMP traffic, which could be generated by the 'snoop' command, which should be identified and subsequently logged. As per the specification, ICMP traffic is not essential so can in theory be blocked on the firewall for botnet activity prevention. The last rule defined will check for a [SYN], [ACK] packet sent from any IP using port 5001, as that is the port that the botnet uses to communicate, and the packet identified is a part of the handshake used to establish the communication link between controller and bot. If this could be further analysed to see if the following packets (specifically [PSH], [ACK] packets) contain one of a select amount of keywords, it could be dropped at the router to disable botnet activity.

Proof of the IDS rules working can be found in figures 29 and 30. Figure 29 shows botnet activity on the port being detected, regular botnet activity being flagged, and the botnet handshake being identified. Figure 30 shows the ICMP traffic being flagged as per the ruleset in figure 28.

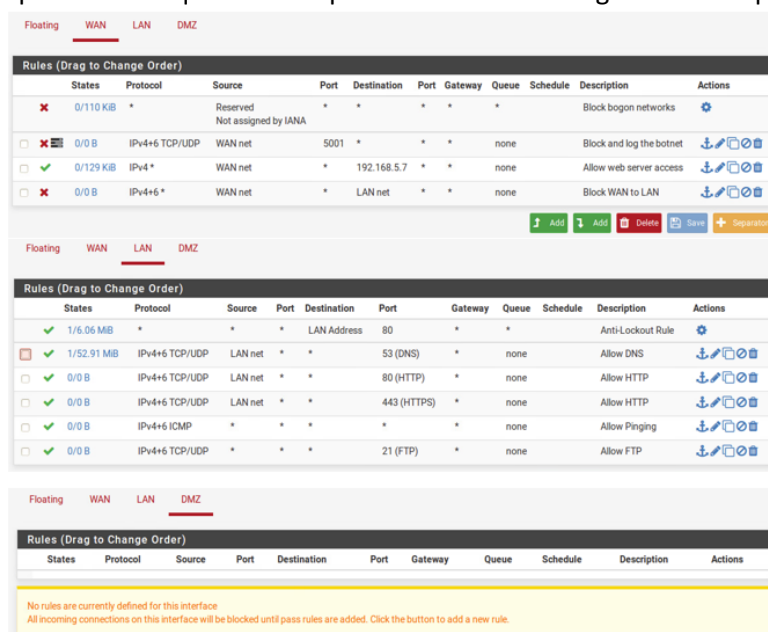


(figure 29) alerts generated from IDS rules



(figure 30) alerts generated from IDS rules

From this, it would be rational to create a closed stance firewall configuration to prevent future communications for this particular botnet agent and controller, but allow certain valid traffic, as specified in requirements specification. This configuration and proof can be seen in figures 31 – 34.

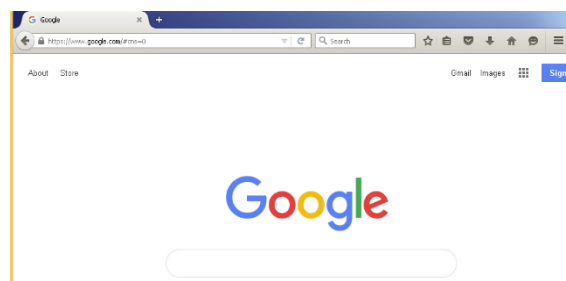


(figure 31) Pfsense firewall setup to meet the specification

The figures show the rules applied to the relevant interfaces on the firewall to enable the closed stance that is required by the specification. The descriptions found on each should be subsequent to explain the purpose of each rule. Following this, an adequate explanation of each can be found. The main purpose of the rules was to prevent the botnet traffic from entering the network, which is shown as evidence below

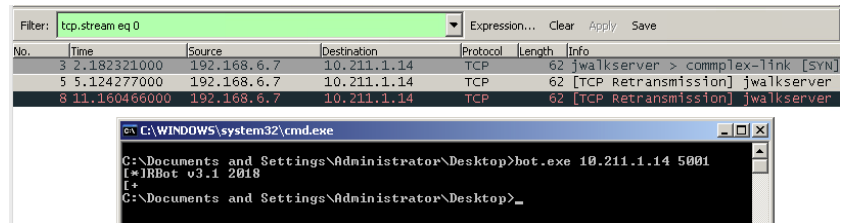
As required, the rules on the WAN interface are responsible for stopping any IP using port 5001 to enter the network as the controller uses 5001 to communicate. As well as this, the WAN rules are in place to allow web access to the appropriate IP only, as there is a following block all rule.

In addition, the LAN rules in place are there to meet the spec, as well as allow regular connections on ports 80, 443, 53, 21 (HTTP(S), DNS and FTP) as well as allow pinging internally, but not externally as per the block on the WAN interface.

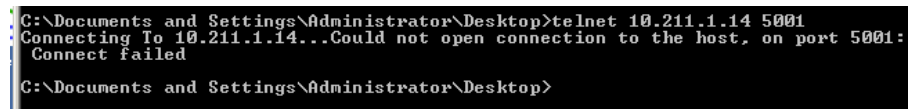


(figure 32) Internal network to HTTPS

As can be seen in figures 32 - 34, the firewall rules, having been applied, are stopping the botnet from working as the bot application cannot establish a connection to the controller, due to the rules on the WAN interface stopping the normal SYN ACK packets being received on the internal network. This is proven in both the Wireshark window and the bot application not establishing a connection, as there is no return SYN ACK packet from



(figure 33) Wireshark capture of connection failure



(figure 34) Botnet failing to connect to controller

the destination address, shown in Wireshark, and the bot does not run any commands, but closes instead. Figure 34 below shows that even the telnet option to send commands to the controller do not work, as a connection from port 5001 cannot be established. This is due to the firewall rules working as intended.

In relation to allowing regular traffic as per the specification, figure 32 shows how the internal network still has internet access in accordance with the rules in place, proving SYN ACK packets are still allowed from the outside internet as these packets are part of any handshake which is required to establish an online connection, providing they are not originating from port 5001.

The prevention of this connectivity will ensure that the internal network cannot communicate with the botnet software used, so the activities that the bot runs when active will not occur on the network. For example, the 'snoop' command's activity in which ARP / ICMP traffic is sent to every IP in the last octet of the local address will not occur, so would not require a firewall rule, which subsequently reduces the overhead on the router as well as allow the internal network to utilise genuine ICMP / ARP packets as would be expected on a normal network in both testing and production scenarios.

## Section 4 - References

### Bibliography

Aussems, E., Noë, B. & Rivera, N., 2014. *Botnets - A tenacious Web Technology*, Leiden: Leiden University.

Chawla, A., Lee, B., Fallon, S. & Jacob, P., 2018. *Host based Intrusion Detection System with*, Ireland: Athlone Institute of Technology.

Cisco, 2019. *Snort - Network Intrusion Detection & Prevention System*. [Online]  
Available at: <https://www.snort.org/>  
[Accessed 9 April 2019].

CrackStation, 2019. *CrackStation*. [Online]  
Available at: <https://crackstation.net/>  
[Accessed 9 April 2019].

GCHQ, 2019. *CyberChef*. [Online]  
Available at: <https://gchq.github.io/CyberChef/>  
[Accessed 9 April 2019].

Kaspersky, 2019. *What is Botnet? | Preventing Botnet Attacks | Kaspersky Lab UK*. [Online]  
Available at: <https://www.kaspersky.co.uk/resource-center/threats/botnet-attacks>

Oracle, 2019. *Oracle VM VirtualBox*. [Online]  
Available at: <https://www.virtualbox.org/>  
[Accessed 9 April 2019].

Panda Security, 2019. *What is a Botnet and How does it work? -*. [Online]  
Available at: <https://www.pandasecurity.com/mediacenter/security/what-is-a-botnet/>  
[Accessed 9 April 2019].

The Honeynet Project, 2008. *The Honeynet Project, 2008. Uses of Botnets*. [Online]. [Online]  
Available at: [Available at: https://www.honeynet.org/node/52](https://www.honeynet.org/node/52)  
[Accessed 14 March 2019].

VMware, 2019. *VMware vSphere*. [Online]  
Available at: <https://vsoc.napier.ac.uk/vsphere-client/?csp>  
[Accessed 9 April 2019].

Vuong , S. & Alam , M., 2011. Advanced Methods for Botnet Intrusion Detection Systems. *Intrusion Detection Systems*, pp. 55-80.

Wireshark, 2019. *Wireshark · Go Deep..* [Online]  
Available at: <https://www.wireshark.org/>  
[Accessed 10 April 2019].

# Appendix

## Appendix A

Floating

WAN

LAN

DMZ

Rules (Drag to Change Order)

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<div>✖</div> 0/39 KiB	*	Reserved Not assigned by IANA	*	*	*	*	*		Block bogon networks	<div>⚙</div>
<div>📄</div> <div>✔</div> 0/0 B	IPv4 *	*	*	*	*	*	none			<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>
<div>📄</div> <div>✔</div> 0/0 B	IPv6 *	*	*	*	*	*	none			<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>

Floating

WAN

LAN

DMZ

Rules (Drag to Change Order)

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<div>✔</div> 1/2.37 MiB	*	*	*	LAN Address	80	*	*		Anti-Lockout Rule	<div>⚙</div>
<div>📄</div> <div>✔</div> 1/1.72 MiB	IPv4 *	LAN net	*	*	*	*	none		Default allow LAN to any rule	<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>
<div>📄</div> <div>✔</div> 0/0 B	IPv6 *	LAN net	*	*	*	*	none		Default allow LAN IPv6 to any rule	<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>

Floating

WAN

LAN

DMZ

Rules (Drag to Change Order)

States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<div>📄</div> <div>✔</div> 0/0 B	IPv6 *	*	*	*	*	*	none			<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>
<div>📄</div> <div>✔</div> 0/0 B	IPv4 *	*	*	*	*	*	none			<div>🔗</div> <div>✎</div> <div>📄</div> <div>🗑</div>

## Appendix B

- i. "The distributed denial of service attacks against dynamic domain name service provider Dyn this morning have now resurged. The attacks have caused outages at services across the Internet."
- ii. "The botnet, made up of devices like home Wi-Fi routers and Internet protocol video cameras, is sending massive numbers of requests to Dyn's DNS service. Those requests look legitimate, so it's difficult for Dyn's systems to screen them out from normal domain name lookup requests."
- iii. "An advanced piece of malware, known as Regin, has been used in systematic spying campaigns against a range of international targets since at least 2008. A back door-type Trojan, Regin is a complex piece of malware whose structure displays a degree of technical competence rarely seen. Customizable with an extensive range of capabilities depending on the target, it provides its controllers with a powerful framework for mass surveillance and has been used in spying operations against government organizations, infrastructure operators, businesses, researchers, and private individuals."
- iv. "Hundreds of thousands of Android phones have been infected with malware that uses handsets to send spam and buy event tickets in bulk. Mobile security firm Lookout said the virus, called NotCompatible, was the most sophisticated it had seen. The cyberthieves behind it had recently rewritten its core code to make it harder to defeat, it said. Mobile malware aimed at smartphones is steadily getting more complex, said security company Wandera. Jeremy Linden, a security analyst at Lookout, said: \"The group behind NotCompatible are operating on a different plane to the typical mobile malware maker.\""
- v. "Prince added that Cloudflare was seeing a sizable increase in errors in traffic for its customers because the attack was affecting infrastructure providers like GitHub. \"If a customer's site is pointing to a git there, now we can't reach Github,\" he said. \"There are definitely infrastructure providers that we can't reach.\""



## Appendix C

Recipe

From Base64

Alphabet  
A-Za-z0-9+/=

☒ Remove non-alphabet chars

Render Image

Input format  
Raw

Input

length: 41277  
lines: 2

kpXp5tLum4lyr0dz/Lp1wJ2cXCKn6mW6t65QhtpCnAlkqslKQzibw9cQ6ghr1Kxgs+dV8l1JQ69y0qIuFdv3w1V5A8ghN8smypPbQlXv1o0gtqkSpCFp580YfNr3Iubb+oxAS42kAKQEGkqTbSEHdvocEF88d0C/fy0/PA0rPmxdM0uau121Ap9bIsh8VX0Yp2kQg0i1vhdH8Avp2V3g4pQ5TV54fKXUN9P+u8KQZweh57C91b177YfCfR1Hm5a1nhdS0f519eLumAtPlRdaymxvQqem/rfpgos1TQXqlyp1CAKFDtcjsfBALR3jns5t0rnoRkQkTF6TFH32nfDqskJanf1zSKAS49K4a0eQp70Pmpseid1A30a87Ygvo1nWp1JUSC5K+eF9a9/r1j4gKL10H7KzJ8ku25+10919BmQp4d13k6vy7Fvu0BdAEwVzAtC1a13312e1pXK9a1JX8qW4/T3nrK+AcChntrgTSkld6XvGatrgy0d5aVEgtGxtdKX1kDR9vaggUlgktrTvocg6g7r9KcckawvYhNHCUp6bVXAL1JoktdawoQkF7naZVoVqg10p9Hq3635HBA20C1SUKCFnsXpYfna+Chu8BF+SkpPyoEq6X02KESnW0J1+78KkQkTq1eYl907415pPcb1IBkhe9wR9L411rPrUBUICXUgBKE+zbzfn0vgJolyREoupcakS18L6R5rSvshUjoeFnA015ACutU0OrJ56sa9hsb7khw8AZuALc7LXvc7B10n1/LB590+ntU1KvMlyk26+xxQ41C3J5AN04ns+73P7W8Zwy+ZqLduK/zwZ0N2ZLcbbjqJdDB35K+1gcVRQ6SMA+1Ry+as3cc1qlpssp/UA2P5JAFht101LBurQd1Ncx7JAKaao30J2C5SLU5o673wCcxp8Np5qd1pu1yog9N+50Cyg+Y3Jrvt11T5nbtJk8bPhfr1/PBFcm1urcxKUTXnd32SHLtbKtP88KkyH/PF0SHQc8CFWQzSHCB+6EfnnHFPK11n3KyzBU1pa2qnLbcVQ6t5NvMPl/3Y31TUnUf9+RhhczVd0F1bKcVSLTcbqJ1N7EXHkZncvdmZHz75Kkuuy29avKCN0r/D2x12dnpK9Sul15m9KCS1Hqb3wCvXpDhC1Nwyvd8tvgvAUu0h1p1an2g1TVUuv0M936c5VJLbV1NkrTydQfLi0u3pF78QufH8bp7enXAK10aj3yZrbfprul/A34qgy0UUTJQ5HQ8LmbD5d26414g3Tq5CmYl1iu/pg08sh55UgD5/Ub2HFF0TAlnv52038mUj5d02Ahop8UpQuD297Af6/TEAyH8C2LkWhVYC/Y3u0H+JwB45Um1BCKbW/v4vqWKCdK21pccpCR0duh2+e+2A6Ehcn5K4pJp27w+K/09/pg01TR1Uu6n6BV1x022PmgdJ/LCEB1HVPF+v8Aaastbba3k0bCVULFkgU/WB2xPOLv0h3YH0KkL1J2AQdF+v0u4545HkXSnoukVuf19E12ATG0RCR18W7At1p1IIEg8gpkct89Bksn23G0MPSH3UBvUFv2nngD89/r18u1fW0XgU/0pDq1+AAU0A1KkHkFKb+u385FFv5PuyL1J3+thbvgK5z1+X3DYS1dLV1EA2K10x+V+15EQ01bplvnyw7J3m1.2V2Vv1nH5cLw8r21Zu1C9KQ1radQKFFe251c11eTqCwKdHkxvYejJ5shoa3H5d1h1bbR4J95tF45QJkxudscKYeqK5QeUjXhCLqME2sre2PgJ5HQ8FHW3K8235JOPN3/LBTU2JYjnmV5Kw4ufRammQKQ69+u6EYJ1e5ht1J5ny/d3810r678Zb18bb47ag5hZa+ZQ3hZftgC5vqI6t+YTTvR0wK5u3cE406JmU/d207u/h1E6z1N74g4y08tptAD54p0vex7A77fU0cUcctbba5p5Q040v91TVIGKrqvE54V51d+vnlmrt1BYv0t0em09jYEA97X8sdxADYTDK11XQcKxuoDrF0wQ54bnQnpbUghs18VYK5Q4+0J1IR8u0+0015A/LBCduKp3jpkR5D80y09uEVhK0hplh5yvytFHWB72177Df8Reh58s0k3VulKn5074DK11Q4pQCCnz15np6BQ63u8R4u1dRCukr1DYL1n+3m7Y10CPHFNfkaE/rKUR5800y0c61638cJpc3N7Hf06YqH1142HXY7+VH4H0Hv/ZQ==

Output

length: 2095  
lines: 189

y0yà...JFIF....H..H...y0.C.....

## Appendix D

Recipe

From Base64

Alphabet  
A-Za-z0-9+/=

☒ Remove non-alphabet chars

Render Image

Input format  
Raw


Input

length: 41277  
lines: 2

kpXp5tLum4lyr0dz/Lp1wJ2cXCKn6mW6t65QhtpCnAlkqslKQzibw9cQ6ghr1Kxgs+dV8l1JQ69y0qIuFdv3w1V5A8ghN8smypPbQlXv1o0gtqkSpCFp580YfNr3Iubb+oxAS42kAKQEGkqTbSEHdvocEF88d0C/fy0/PA0rPmxdM0uau121Ap9bIsh8VX0Yp2kQg0i1vhdH8Avp2V3g4pQ5TV54fKXUN9P+u8KQZweh57C91b177YfCfR1Hm5a1nhdS0f519eLumAtPlRdaymxvQqem/rfpgos1TQXqlyp1CAKFDtcjsfBALR3jns5t0rnoRkQkTF6TFH32nfDqskJanf1zSKAS49K4a0eQp70Pmpseid1A30a87Ygvo1nWp1JUSC5K+eF9a9/r1j4gKL10H7KzJ8ku25+10919BmQp4d13k6vy7Fvu0BdAEwVzAtC1a13312e1pXK9a1JX8qW4/T3nrK+AcChntrgTSkld6XvGatrgy0d5aVEgtGxtdKX1kDR9vaggUlgktrTvocg6g7r9KcckawvYhNHCUp6bVXAL1JoktdawoQkF7naZVoVqg10p9Hq3635HBA20C1SUKCFnsXpYfna+Chu8BF+SkpPyoEq6X02KESnW0J1+78KkQkTq1eYl907415pPcb1IBkhe9wR9L411rPrUBUICXUgBKE+zbzfn0vgJolyREoupcakS18L6R5rSvshUjoeFnA015ACutU0OrJ56sa9hsb7khw8AZuALc7LXvc7B10n1/LB590+ntU1KvMlyk26+xxQ41C3J5AN04ns+73P7W8Zwy+ZqLduK/zwZ0N2ZLcbbjqJdDB35K+1gcVRQ6SMA+1Ry+as3cc1qlpssp/UA2P5JAFht101LBurQd1Ncx7JAKaao30J2C5SLU5o673wCcxp8Np5qd1pu1yog9N+50Cyg+Y3Jrvt11T5nbtJk8bPhfr1/PBFcm1urcxKUTXnd32SHLtbKtP88KkyH/PF0SHQc8CFWQzSHCB+6EfnnHFPK11n3KyzBU1pa2qnLbcVQ6t5NvMPl/3Y31TUnUf9+RhhczVd0F1bKcVSLTcbqJ1N7EXHkZncvdmZHz75Kkuuy29avKCN0r/D2x12dnpK9Sul15m9KCS1Hqb3wCvXpDhC1Nwyvd8tvgvAUu0h1p1an2g1TVUuv0M936c5VJLbV1NkrTydQfLi0u3pF78QufH8bp7enXAK10aj3yZrbfprul/A34qgy0UUTJQ5HQ8LmbD5d26414g3Tq5CmYl1iu/pg08sh55UgD5/Ub2HFF0TAlnv52038mUj5d02Ahop8UpQuD297Af6/TEAyH8C2LkWhVYC/Y3u0H+JwB45Um1BCKbW/v4vqWKCdK21pccpCR0duh2+e+2A6Ehcn5K4pJp27w+K/09/pg01TR1Uu6n6BV1x022PmgdJ/LCEB1HVPF+v8Aaastbba3k0bCVULFkgU/WB2xPOLv0h3YH0KkL1J2AQdF+v0u4545HkXSnoukVuf19E12ATG0RCR18W7At1p1IIEg8gpkct89Bksn23G0MPSH3UBvUFv2nngD89/r18u1fW0XgU/0pDq1+AAU0A1KkHkFKb+u385FFv5PuyL1J3+thbvgK5z1+X3DYS1dLV1EA2K10x+V+15EQ01bplvnyw7J3m1.2V2Vv1nH5cLw8r21Zu1C9KQ1radQKFFe251c11eTqCwKdHkxvYejJ5shoa3H5d1h1bbR4J95tF45QJkxudscKYeqK5QeUjXhCLqME2sre2PgJ5HQ8FHW3K8235JOPN3/LBTU2JYjnmV5Kw4ufRammQKQ69+u6EYJ1e5ht1J5ny/d3810r678Zb18bb47ag5hZa+ZQ3hZftgC5vqI6t+YTTvR0wK5u3cE406JmU/d207u/h1E6z1N74g4y08tptAD54p0vex7A77fU0cUcctbba5p5Q040v91TVIGKrqvE54V51d+vnlmrt1BYv0t0em09jYEA97X8sdxADYTDK11XQcKxuoDrF0wQ54bnQnpbUghs18VYK5Q4+0J1IR8u0+0015A/LBCduKp3jpkR5D80y09uEVhK0hplh5yvytFHWB72177Df8Reh58s0k3VulKn5074DK11Q4pQCCnz15np6BQ63u8R4u1dRCukr1DYL1n+3m7Y10CPHFNfkaE/rKUR5800y0c61638cJpc3N7Hf06YqH1142HXY7+VH4H0Hv/ZQ==

Output

length: 2095  
lines: 189



## Appendix E

https://www.dcode.fr/vigenere-cipher

Search for a tool

★ SEARCH A TOOL ON DCODE BY KEYWORDS:

e.g. type caesar

GO

Results

11	11
5 lett.	■■■■■■■
10 lett.	■■■■■■
15 lett.	■■■■■■■
2 lett.	■■■■■
4 lett.	■■■■
8 lett.	■■■■
16 lett.	■■■■
3 lett.	■■■■
14 lett.	■■■
18 lett.	■■■
17 lett.	■■■
11 lett.	■■■
13 lett.	■■■
12 lett.	■■■
9 lett.	■■■
7 lett.	■■■
6 lett.	■■■
19 lett.	■■■
#18	

VIGENERE CIPHER

Cryptography > Poly-Alphabetic Cipher > Vigenere Cipher

Sponsored ads

Try Google Advertising

When people search online, make sure they find you – with Google Ads

ads.google.com

OPEN

Vigenere Decoder

★ VIGENERE CIPHERTEXT

LRO LTVGASH/DICIGIXY RPXIRPH- PRSAPGNN KLWT CJXFEHGH ZJ

INTDI QIKTIGEH XYXO P OZANTIA- BLRW IM TWY YATS ES

CSQSCQI SOSF PEECH- QPAMAKK ELB DUTZGPH TWFE ET

ATWIT DIXI OM ISI QIKTIGEH ZDIO XC ELE SNY HNN PEXARZD ERT

QWVS- UQCHTIS ZETKWTG IWP XERWYXCPA TSOXRLXOGU LEO

IZVYRHH- EICQCUUVP- LEO KQWESYKIS PHSCKNFEAD LKSL

PATGMQJH VROLC XMPX MSICTE EITPNOS-

KNOWING THE KEY: APPLE

KNOWING THE KEY-LENGTH, SIZE: 5

KNOWING ONLY A PARTIAL KEY: KE?

KNOWING A PLAINTEXT WORD: CODE

TRY A COMMON-WORDS DICTIONARY ATTACK

TRY TO DECRYPT AUTOMATICALLY (STATISTICAL ANALYSIS)

★ ALPHABET ABCDEFGHIJKLMNOPQRSTUVWXYZ

DECRYPT VIGENERE

→ Beaufort Cipher — Caesar Cipher

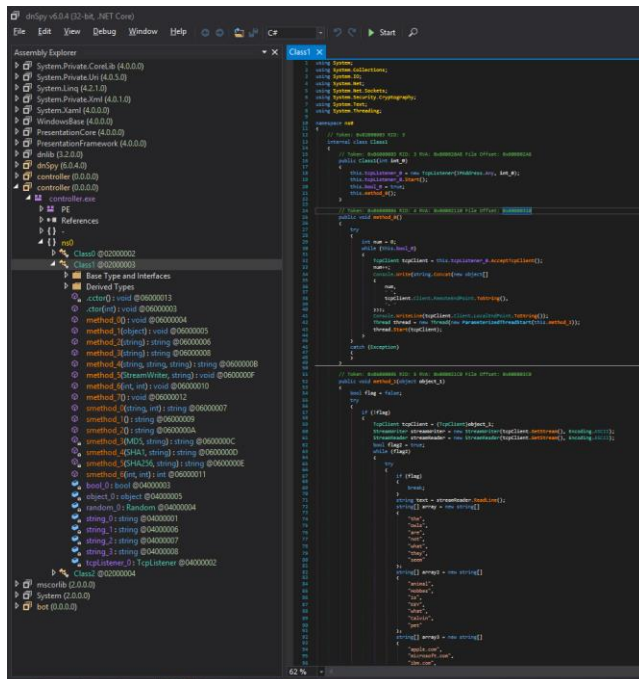
Vigenere Cryptanalysis

★ KASISIK'S TEST

FIND KEY LENGTH

Page 19 of 20

## Appendix F



## Appendix G

