Analysis of a Botnet's impact on a network and possible countermeasures

Tommaso Liardo
40452307
40452307@napier.ac.uk
Edinburgh Napier University - Network Security and Cryptography (CSN09112)

December 17, 2021

Summary

1 Introduction

A botnet is a network of compromised, remotely controlled computer systems created with malicious purposes such as distribution of spam e-mails, coordination of DDoS attacks and automated identity theft (Plohmann *et al*, 2011).
The list of the possible purposes can be further expanded by adding social media attacks (such as Twitter/Facebook spam) and cryptocurrency scams.

Botnets are created by infecting a large number of devices through techniques like popup ads, email attachment, downloads from the web. The “masters” creating and controlling the botnet do not look for specific individuals or companies but target any device vulnerable enough to fall for the trap and become infected and thus enlarging their bot network.

The spirit of the botnet system is to use a small fraction of the processing power from the sheer number (often hundreds of thousands) of hosts infected rather than taking complete control over a few devices, as the latter will alert the users of malicious activities that they will try to stop. Instead, the goal of hackers who create botnet is to make them as large and durable as possible, so infected devices will often never notice that they’re part of the network.

The aim of this paper is to discuss the following points:

1. Description of the starting configuration, the tools and methods used to perform the analysis.
2. Analysis of the operations of the Bot agent and Botnet controller inside the network, including connections/communications between the two, host activities on the victim's devices, and any other possible behaviour connected to the Botnet's activity while highlighting the tools and methods used.
3. Attempts of possible countermeasures to a Botnet attack and the prototype of a detection system for future attacks. For instance, the implementation of an IDS system prototype using Snort and a valid configuration/set of rules in the firewall.

A network topology has been configured as a test bed for this purpose.

2 Network Configuration and Tools

The network topology for this analysis is the following:

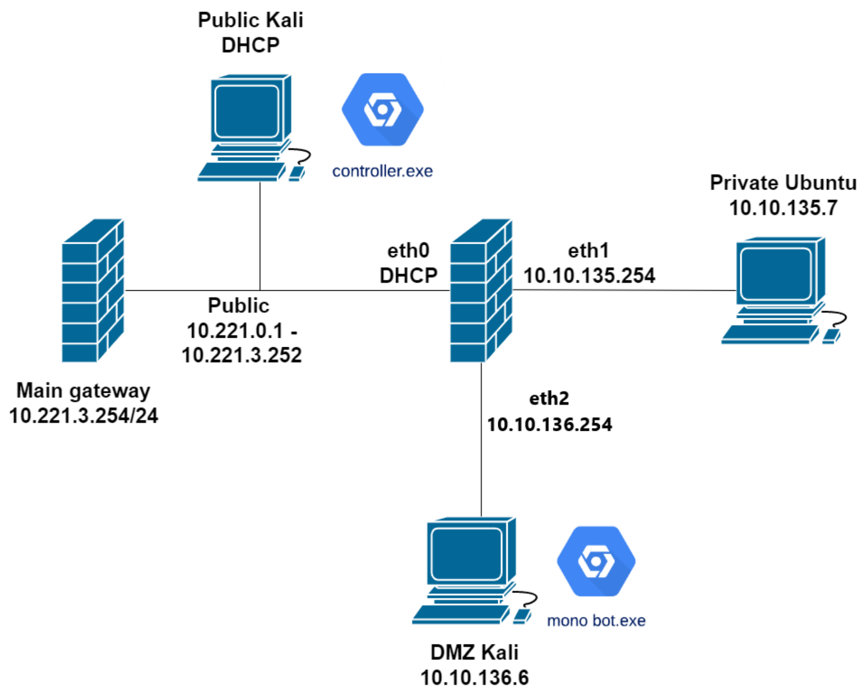


Figure 1. Network Topology

As highlighted in Figure 1, the *controller.exe* file is located in the Kali machine inside the public network, and the *mono bot.exe* file is in the Kali machine inside the DMZ. The IP address assigned by the DHCP to the public Kali device during this test is 10.221.0.247.
NAT has been set up to allow communications from the private and DMZ networks.
Basic firewall rules have been set up.

2.1 NAT Configuration

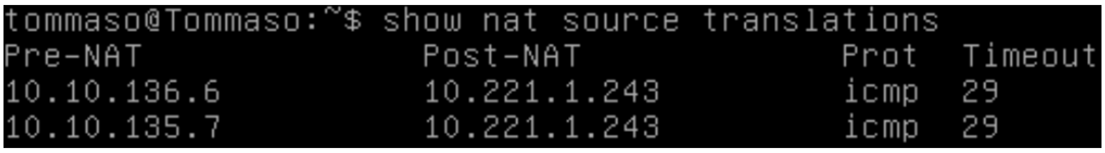
Mapping the addresses from private network to the public network:

```
set nat source rule 1 outbound-interface eth0
set nat source rule 1 source address 10.10.135.0/24
set nat source rule 1 translation address masquerade
```

Mapping the addresses from the DMZ network to the public network:

```
set nat source rule 2 outbound-interface eth0
set nat source rule 2 source address 10.10.136.0/24
set nat source rule 2 translation address masquerade
```

Result from the Vyatta firewall:



```
tommaso@Tommaso:~$ show nat source translations
Pre-NAT              Post-NAT              Prot   Timeout
10.10.136.6          10.221.1.243          icmp   29
10.10.135.7          10.221.1.243          icmp   29
```

2.2 Firewall Configuration

Creating the public network zone:

```
set zone-policy zone public description "Outside"
set zone-policy zone public interface eth0
```

Creating the private network zone:

```
set zone-policy zone private description "Inside"
set zone-policy zone private interface eth1
```

Creating the DMZ zone:

```
set zone-policy zone dmz description "DMZ"
set zone-policy zone dmz interface eth2
```

Since the DMZ is created to act as a buffer between the private network and the public network to add another layer of security to the private network, it will only accept connections on port 80 and 443 (HTTP and HTTPS protocol) from the private network, and the private network will only accept established connections from the DMZ.

Private to DMZ network:

```
set firewall name private2dmz description "private to DMZ"
set firewall name private2dmz rule 1 action accept
set firewall name private2dmz rule 1 state established enable
set firewall name private2dmz rule 1 state related enable
set firewall name private2dmz rule 10 action accept
set firewall name private2dmz rule 10 destination port 80,443
set firewall name private2dmz rule 10 protocol tcp
set zone-policy zone dmz from private firewall name private2dmz
```

DMZ to private network:

```
set firewall name dmz2private description "DMZ to private"
set firewall name dmz2private rule 1 action accept
set firewall name dmz2private rule 1 state established enable
set firewall name dmz2private rule 1 state related enable
set zone-policy zone private from dmz firewall name dmz2private
```

Private to public network:

```
set firewall name private2public description "Private to public"
set firewall name private2public rule 1 action accept
set zone-policy zone public from private firewall name private2public
```

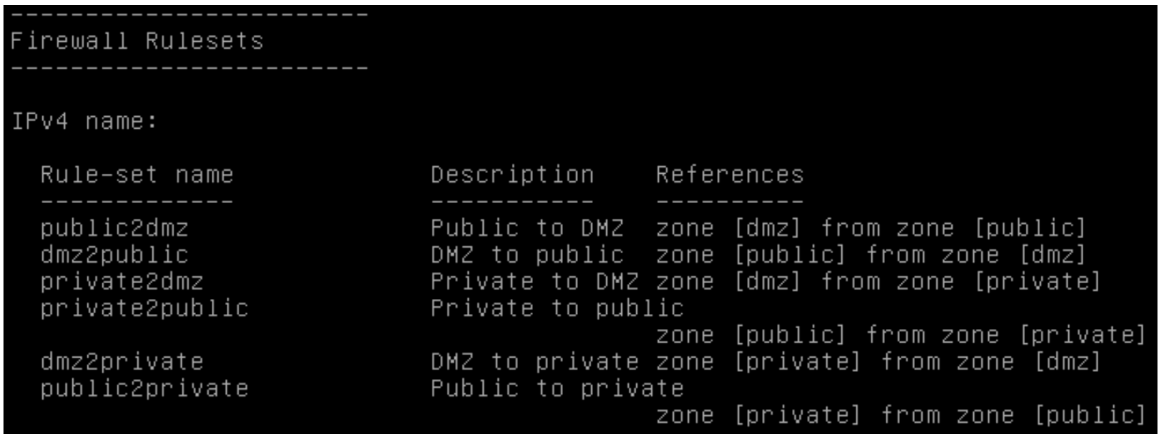
Public to private network:

```
set firewall name public2private description "Public to private"
set firewall name public2private rule 1 action accept
set firewall name public2private rule 1 state established enable
set firewall name public2private rule 1 state related enable
set zone-policy zone private from public firewall name public2private
```

DMZ to public network:
```
set firewall name dmz2public description “DMZ to public”
set firewall name dmz2public rule 1 action accept
set zone-policy zone public from dmz firewall name dmz2public
```

Public to DMZ network:
```
set firewall name public2dmz description “Public to DMZ”
set firewall name public2dmz rule 1 action accept
set firewall name public2dmz rule 1 state established enable
set firewall name public2dmz rule 1 state related enable
set zone-policy zone dmz from public firewall name public2dmz
```

Result from Vyatta firewall:



```
-----------------------
Firewall Rulesets
-----------------------

IPv4 name:

  Rule-set name             Description    References
  -------------             -----------    ----------
  public2dmz                Public to DMZ  zone [dmz] from zone [public]
  dmz2public                DMZ to public  zone [public] from zone [dmz]
  private2dmz               Private to DMZ zone [dmz] from zone [private]
  private2public            Private to public
                                           zone [public] from zone [private]
  dmz2private               DMZ to private zone [private] from zone [dmz]
  public2private            Public to private
                                           zone [private] from zone [public]
```

With this configuration, the private network and the DMZ are able to communicate between each other as long as it is an established connection and both the DMZ and private network are able to communicate with the public network, while the public network is only able to reply to established connections and has no other access to the private and DMZ networks.

2.3 Tools and Methods

For this task, a botnet agent and controller were provided and were located respectively in the DMZ Kali and in the public kali machine. Wireshark was used to capture the traffic on both machines, and snort was used to read and filter the captured traffic based on initial rules, and to later implement additional rules that may identify the presence of botnet connections in the network. Finally, the Vyatta firewall was configured to stop the traffic for this botnet but allowing valid traffic.

The methods for this report follow the steps below:

- Setting up a starting configuration for the network and the systems.
- A few attempts at running the botnet while capturing traffic with Wireshark in the DMZ Kali in order to look for any suspect traffic to focus on as well as other possible bot behaviour and to critically analyse and discuss it.
- Brief suggestion on possible additional filters and firewall configurations to better detect and prevent future communications for this botnet.
- Implementation and testing of the firewall configuration by running the botnet for the third time and analysing the new traffic in both the DMZ Kali and the private Ubuntu.

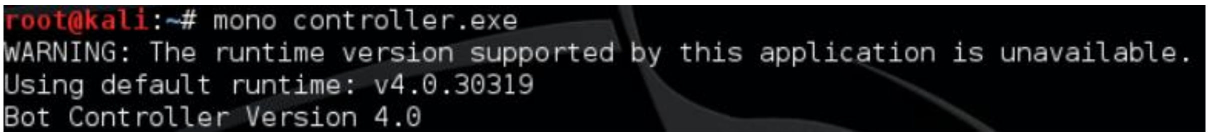
Screenshots of inputs/outputs are provided in each section where relevant.

3 Analysis

The botnet was executed on the DMZ Kali machine using the `"mono botnet.exe 10.221.0.247"` command, while the controller was executed on the public kali machine using `"mono controller.exe"`.

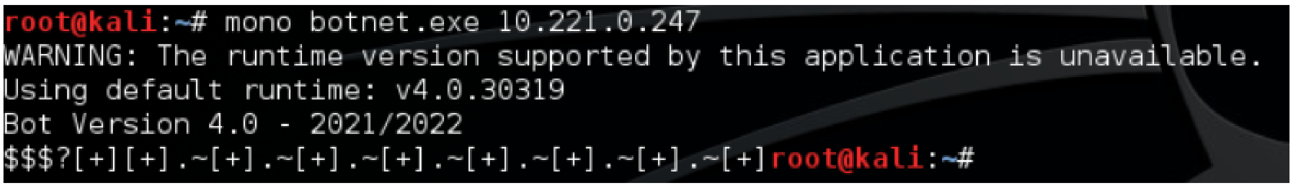
3.1 First Test

After running the botnet controller on the public Kali with the above commands, this was the result:



```
root@kali:~# mono controller.exe
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Controller Version 4.0
```

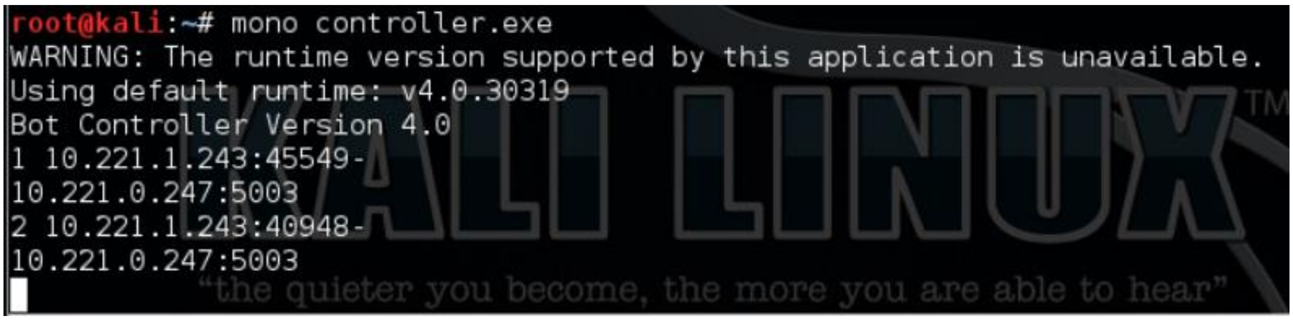
After running the botnet on the DMZ Kali for the first time, this was the result:



```
root@kali:~# mono botnet.exe 10.221.0.247
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$$$?[+][+].~[+].~[+].~[+].~[+].~[+].~[+].~[+]root@kali:~#
```

From a first look it is unclear what these symbols mean, that is why a traffic analysis is crucial to understand how the botnet behaves.

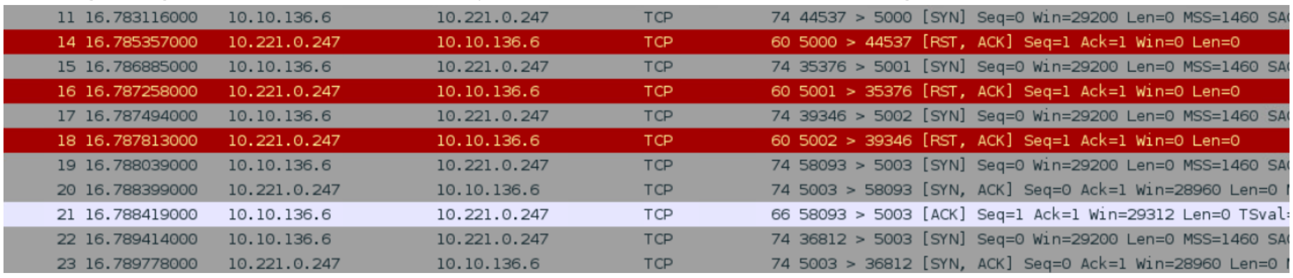
And back to the public Kali:



```
root@kali:~# mono controller.exe
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Controller Version 4.0
1 10.221.1.243:45549-
10.221.0.247:5003
2 10.221.1.243:40948-
10.221.0.247:5003
```

Note that `10.221.1.243` is the DHCP IP address assigned to the Vyatta firewall's interface `eth0`.

The beginning of the traffic stream analysed on the DMZ Kali is the following:

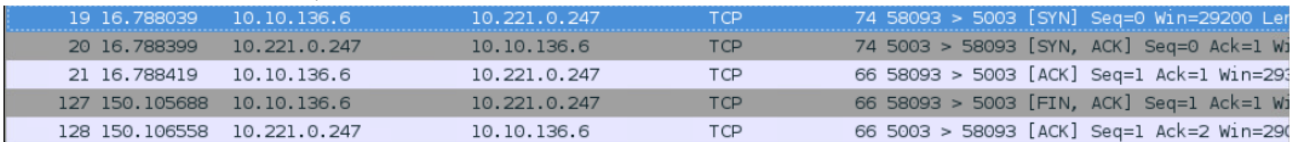


```
11 16.783116000   10.10.136.6    10.221.0.247   TCP   74 44537 > 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
14 16.785357000   10.221.0.247   10.10.136.6    TCP   60 5000 > 44537 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15 16.786885000   10.10.136.6    10.221.0.247   TCP   74 35376 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
16 16.787258000   10.221.0.247   10.10.136.6    TCP   60 5001 > 35376 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17 16.787494000   10.10.136.6    10.221.0.247   TCP   74 39346 > 5002 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
18 16.787813000   10.221.0.247   10.10.136.6    TCP   60 5002 > 39346 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19 16.788039000   10.10.136.6    10.221.0.247   TCP   74 58093 > 5003 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
20 16.788399000   10.221.0.247   10.10.136.6    TCP   74 5003 > 58093 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0
21 16.788419000   10.10.136.6    10.221.0.247   TCP   66 58093 > 5003 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval
22 16.789414000   10.10.136.6    10.221.0.247   TCP   74 36812 > 5003 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
23 16.789778000   10.221.0.247   10.10.136.6    TCP   74 5003 > 36812 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0
```

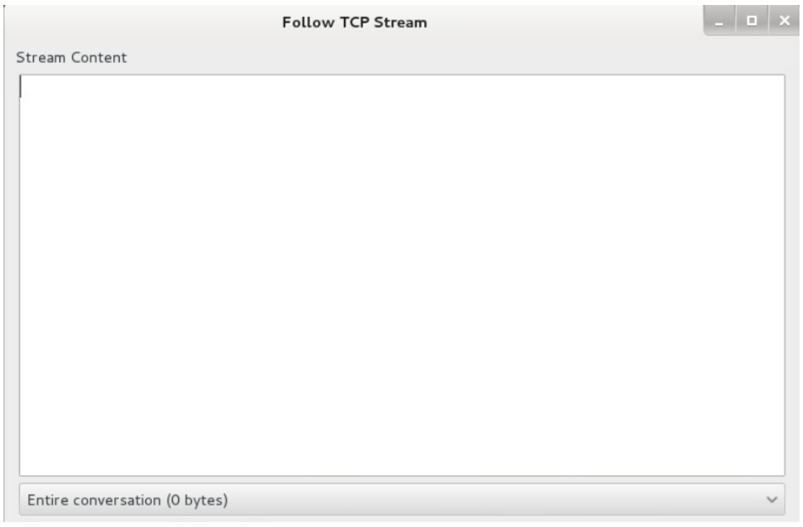
As soon as the command is sent, the DMZ Kali tries to connect to the public Kali on port 5000, but receives a reset reply meaning that the port is most likely closed. Following that, it tries again on port 5001 and port 5002 and finally on 5003 where it receives a `[SYN, ACK]` reply and completes the three-way handshake on that port. This kind of behaviour is suspicious and should alert a security admin as it shows that this machine is trying to reach a remote system on any port until it establishes a connection.

According to IANA, port 5000 is officially registered for the service “*commplex-main*”, port 5001 is officially registered for the service “*commplex-link*”, port 5002 for the service “*rfe*”, and 5003 for the service “*fmpro-internal*”. However, this does not really matter too much, as any port can be used for any service and in fact these ports are associated with different possible services and threats.

Another interesting (and suspicious) behaviour is that after establishing the first connection from por 58093 it does not send any command or other communication.



```
 19 16.788039   10.10.136.6    10.221.0.247   TCP   74 58093 > 5003 [SYN] Seq=0 Win=29200 Ler
 20 16.788399   10.221.0.247   10.10.136.6    TCP   74 5003 > 58093 [SYN, ACK] Seq=0 Ack=1 Wi
 21 16.788419   10.10.136.6    10.221.0.247   TCP   66 58093 > 5003 [ACK] Seq=1 Ack=1 Win=29:
127 150.105688  10.10.136.6    10.221.0.247   TCP   66 58093 > 5003 [FIN, ACK] Seq=1 Ack=1 Wi
128 150.106558  10.221.0.247   10.10.136.6    TCP   66 5003 > 58093 [ACK] Seq=1 Ack=2 Win=29(
```



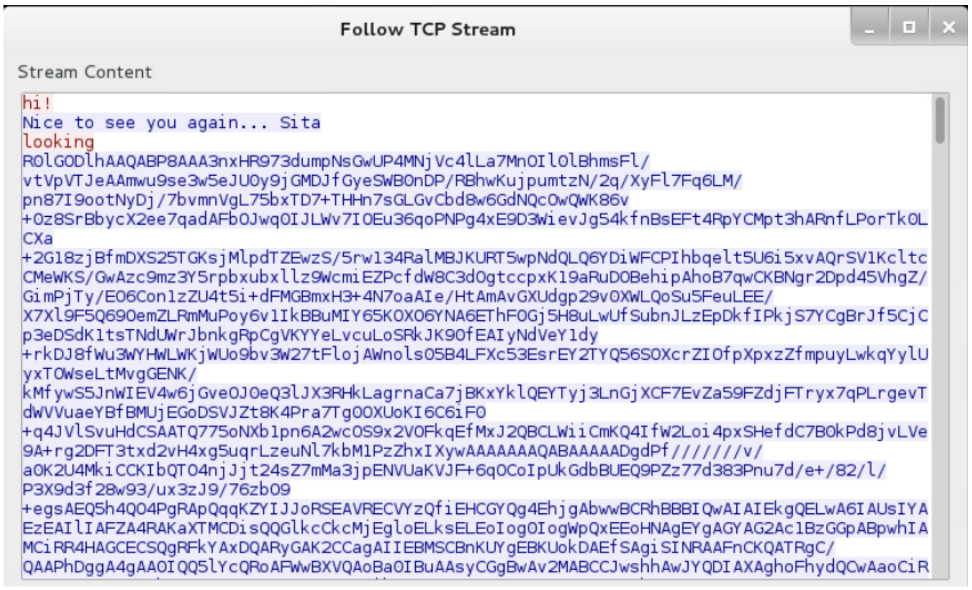
Follow TCP Stream

Stream Content

Entire conversation (0 bytes)

The use of this connection is unclear, but it may be a “decoy” connection created with the intention of leading an admin to think that the connection is safe and nothing dangerous is happening.

Right after establishing that connection, the DMZ Kali establishes a new one (to the same open port it previously discovered) with the controller, but this time the source port is 36812:
Following this stream, a much longer connection is shown (188 packets). The stream content starts as follows:



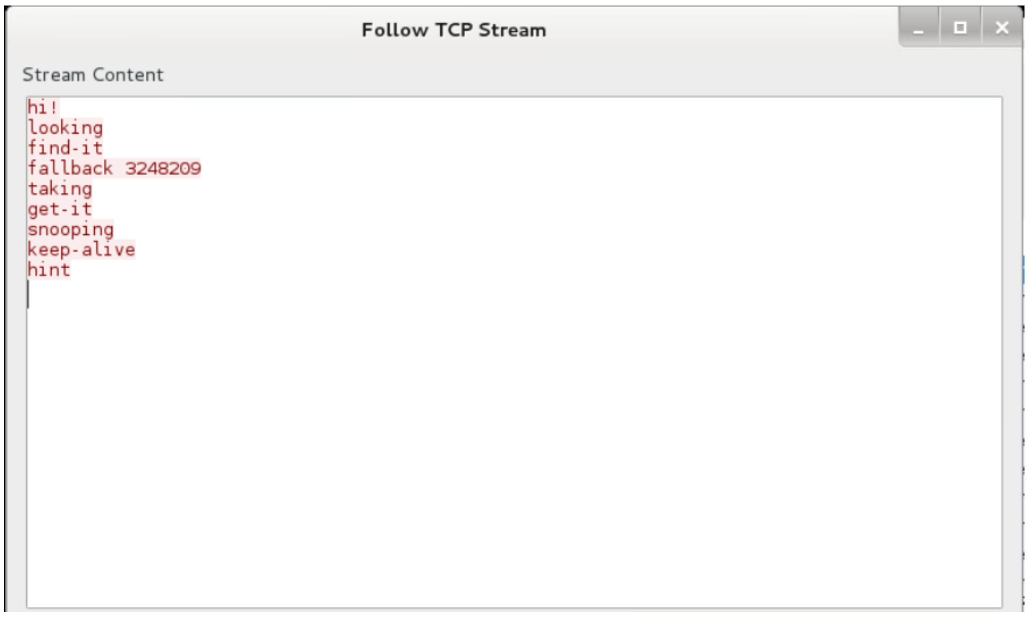
Follow TCP Stream

Stream Content

```
hi!
Nice to see you again... Sita
looking
R0lGODlhAAQABP8AAA3nxHR973dumpNsGwUP4MNjVc4lLa7MnOIl0lBhmsFl/
vtVpVTJeAAmwu9se3w5eJU0y9jGMDJfGyeSWB0nDP/RBhwKujpumtzN/2q/XyFl7Fq6LM/
pn87I9ootNyDj/7bvmnVgL75bxTD7+THHn7sGLGvCbd8w6GdNQcOwQWK86v
+0z8SrBbycX2ee7qadAFbOJwq0IJLWv7IOEu36qoPNPg4xE9D3WievJg54kfnBsEFt4RpYCMpt3hARnfLPorTk0L
CXa
+2G18zjBfmDXS25TGKsjMlpdTZEwzS/5rw134RalMBJKURT5wpNdQLQ6YDiWFCPIhbqelt5U6i5xvAQrSV1Kcltc
CMeWKS/GwAzc9mz3Y5rpbxubxllz9WcmiEZPcfdW8C3d0gtccpxK19aRuDOBehipAhoB7qwCKBNgr2Dpd45VhgZ/
GimPjTy/E06Con1zZU4t5i+dFMGBmxH3+4N7oaAIe/HtAmAvGXUdgp29v0XWLQoSu5FeuLEE/
X7Xl9F5Q690emZLRmMuPoy6v1IkBBuMIY65KOXO6YNA6EThF0Gj5H8uLwUfSubnJLzEpDkfIPkjS7YCgBrJf5CjC
p3eDSdK1tsTNdUWrJbnkgRpCgVKYYeLvcuLoSRkJK90fEAIyNdVeY1dy
+rkDJ8fWu3WYHWLWKjWUo9bv3W27tFlojAWnols05B4LFXc53EsrEY2TYQ56SOXcrZIOfpXpxzZfmpuyLwkqYylU
yxTOWseLtMvgGENK/
kMfywS5JnWIEV4w6jGveOJ0eQ3lJX3RHkLagrnaCa7jBKxYklQEYTyj3LnGjXCF7EvZa59FZdjFTryx7qPLrgevT
dWVVuaeYBfBMUjEGoDSVJZt8K4Pra7Tg0OXUoKI6C6iF0
+q4JVlSvuHdCSAATQ775oNXb1pn6A2wcOS9x2V0FkqEfMxJ2QBCLWiiCmKQ4IfW2Loi4pxSHefdC7B0kPd8jvLVe
9A+rg2DFT3txd2vH4xg5uqrLzeuNl7kbM1PzZhxIXywAAAAAAAQABAAAAADgdPf///////v/
a0K2U4MkiCCKIbQTO4njJjt24sZ7mMa3jpENVUaKVJF+6q0CoIpUkGdbBUEQ9PZz77d383Pnu7d/e+/82/l/
P3X9d3f28w93/ux3zJ9/76zb09
+egsAEQ5h4Q04PgRApQqqKZYIJJoRSEAVRECVYzQfiEHCGYQg4EhjgAbwwBCRhBBBIQwAIAIEkgQELwA6IAUsIYA
EzEAIlIAFZA4RAKaXTMCDisQQGlkcCkcMjEgloELksELEoIog0IogWpQxEEoHNAgEYgAGYAG2Ac1BzGGpABpwhIA
MCiRR4HAGCECSQgRFkYAxDQARyGAK2CCagAIIEBMSCBnKUYgEBKUokDAEfSAgiSINRAAFnCKQATRgC/
QAAPhDggA4gAA0IQQ5lYcQRoAFWwBXVQAoBa0IBuAAsyCGgBwAv2MABCCJwshhAwJYQDIAXAghoFhydQCwAaoCiR
```

The lines in red are from the bot, while the lines in blue are from the controller.

While the messages sent from the controller are mostly encrypted, it was possible to recover the stream content sent from the bot, which is as shown:



```
hi!
looking
find-it
fallback 3248209
taking
get-it
snooping
keep-alive
hint
```

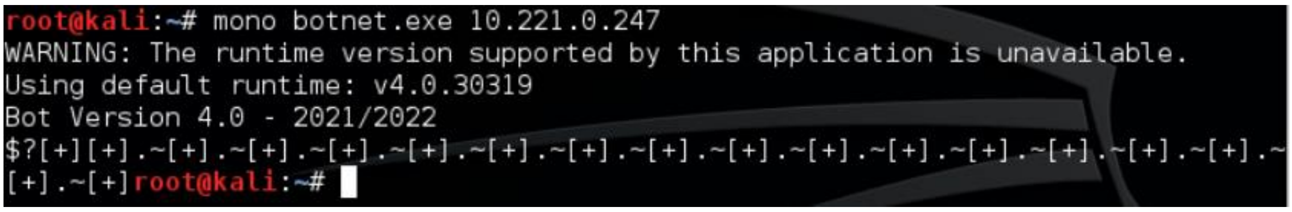
It is unclear what these messages mean as they're likely to be replies to unknown commands from the controller, but there is no evidence so far of any attempt to steal data or to damage the machine, nor any other malicious activity of that kind.

The results of this first test evidence that there is indeed a bot installed and that it is set to try an unknown number of destination ports to establish a connection to its controller every time it is executed, at which point it will receive commands and send replies. The collected data is insufficient to determine its dangerousness for the system or the user.
As bots are likely to be executed repetitively, for the sake of this report it will be run once again to test other potential effects.

During this test there were no apparent signs, nor suspicious behaviours visible on the machine.

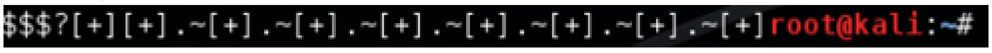
3.2 Second Test

The output on terminal after executing the bot is the following:



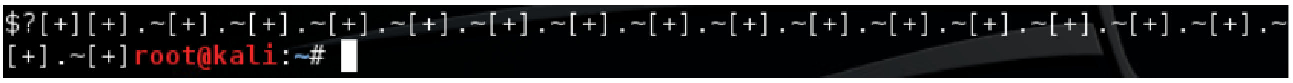
```
root@kali:~# mono botnet.exe 10.221.0.247
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$?[+][+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~
[+].~[+]root@kali:~#
```

This output is different from the first test as shown below.



```
$$$?[+][+].~[+].~[+].~[+].~[+].~[+].~[+].~[+]root@kali:~#
```

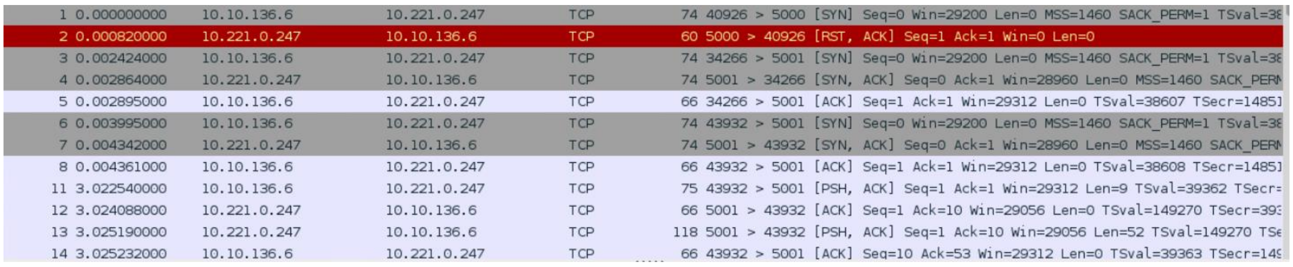
Figure 2. First test.



```
$?[+][+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~[+].~
[+].~[+]root@kali:~#
```

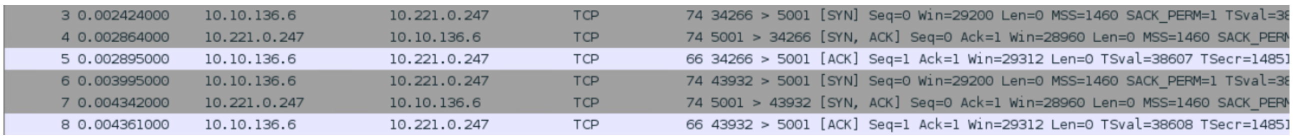
Figure 3. Second test.

The traffic analysed on the DMZ Kali begins as follows:



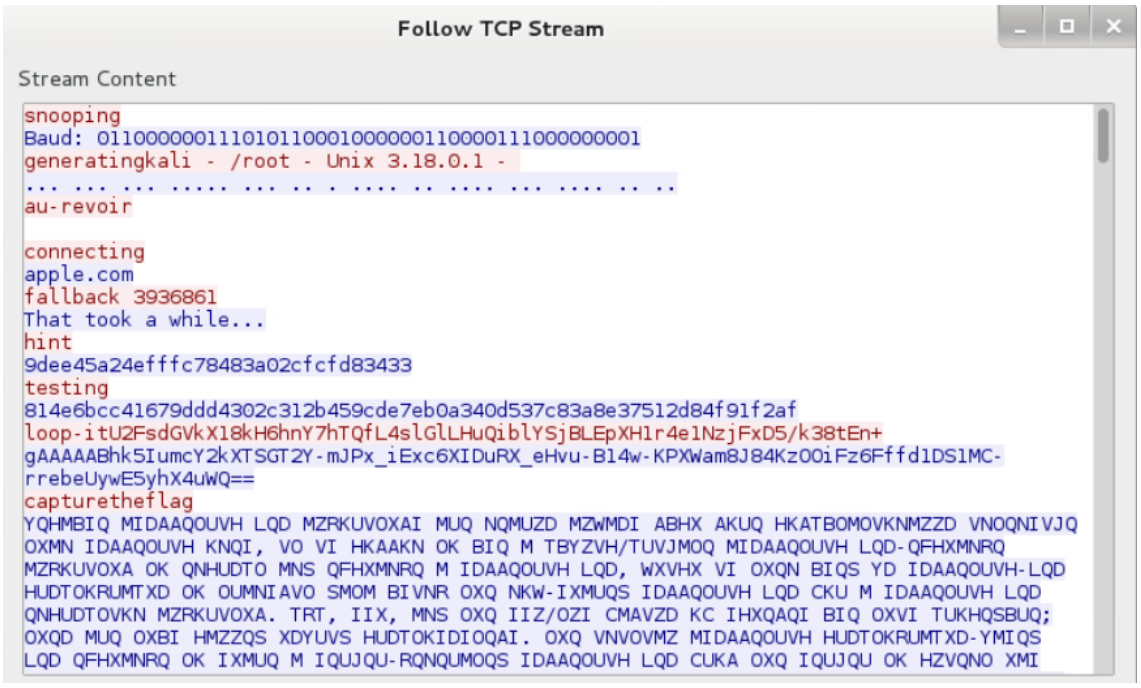
```
 1 0.000000000   10.10.136.6    10.221.0.247   TCP    74 40926 > 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38
 2 0.000820000   10.221.0.247   10.10.136.6    TCP    60 5000 > 40926 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 3 0.002424000   10.10.136.6    10.221.0.247   TCP    74 34266 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38
 4 0.002864000   10.221.0.247   10.10.136.6    TCP    74 5001 > 34266 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM
 5 0.002895000   10.10.136.6    10.221.0.247   TCP    66 34266 > 5001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=38607 TSecr=1485]
 6 0.003995000   10.10.136.6    10.221.0.247   TCP    74 43932 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38
 7 0.004342000   10.221.0.247   10.10.136.6    TCP    74 5001 > 43932 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM
 8 0.004361000   10.10.136.6    10.221.0.247   TCP    66 43932 > 5001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=38608 TSecr=1485]
11 3.022540000   10.10.136.6    10.221.0.247   TCP    75 43932 > 5001 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=9 TSval=39362 TSecr=
12 3.024088000   10.221.0.247   10.10.136.6    TCP    66 5001 > 43932 [ACK] Seq=1 Ack=10 Win=29056 Len=0 TSval=149270 TSecr=39
13 3.025190000   10.221.0.247   10.10.136.6    TCP   118 5001 > 43932 [PSH, ACK] Seq=1 Ack=10 Win=29056 Len=52 TSval=149270 TSe
14 3.025232000   10.10.136.6    10.221.0.247   TCP    66 43932 > 5001 [ACK] Seq=10 Ack=53 Win=29312 Len=0 TSval=39363 TSecr=14
```

As shown, the bot started contacting the controller on port 5000 and the connection was denied and reset on a new port. This time, it was possible to establish a connection on port 5001 and thus the bot did not try to look for more open ports to connect to. Also, just as in the first test, after establishing successfully a connection with the controller, it was left unused and the bot established a second, main connection to communicate changing its source port.



```
3 0.002424000   10.10.136.6    10.221.0.247   TCP    74 34266 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38
4 0.002864000   10.221.0.247   10.10.136.6    TCP    74 5001 > 34266 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM
5 0.002895000   10.10.136.6    10.221.0.247   TCP    66 34266 > 5001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=38607 TSecr=1485]
6 0.003995000   10.10.136.6    10.221.0.247   TCP    74 43932 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38
7 0.004342000   10.221.0.247   10.10.136.6    TCP    74 5001 > 43932 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM
8 0.004361000   10.10.136.6    10.221.0.247   TCP    66 43932 > 5001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=38608 TSecr=1485]
```

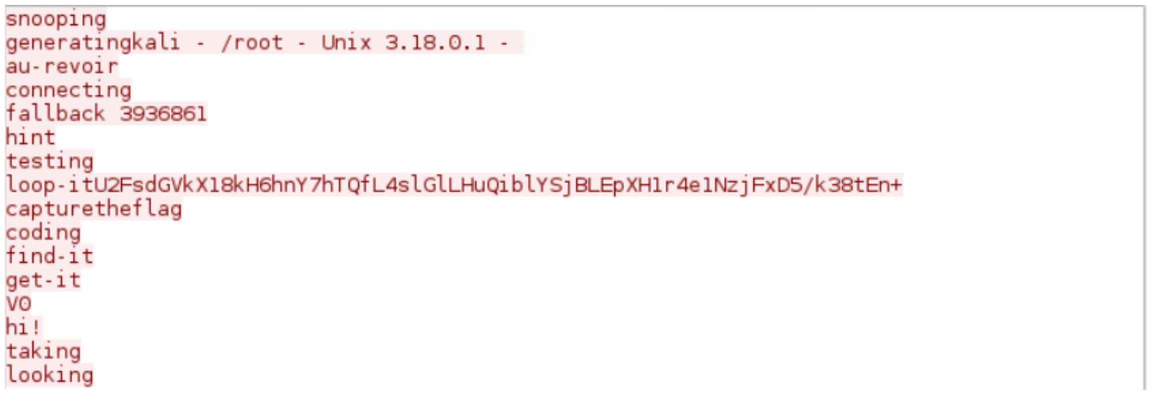
Again, following the main content stream it is possible to see a longer communication (102 packets this time), which is shown below.



Follow TCP Stream

Stream Content

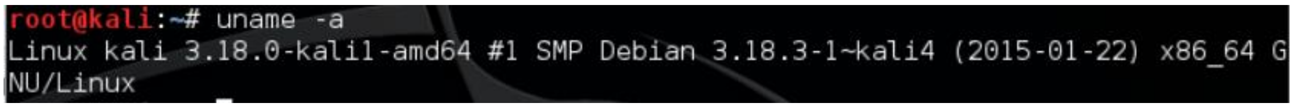
```
snooping
Baud: 011000000111010110001000000110000111000000001
generatingkali - /root - Unix 3.18.0.1 -
... ... ... ..... ... .. . .... .. .... ... .... .. ..
au-revoir

connecting
apple.com
fallback 3936861
That took a while...
hint
9dee45a24efffc78483a02cfcfd83433
testing
814e6bcc41679ddd4302c312b459cde7eb0a340d537c83a8e37512d84f91f2af
loop-itU2FsdGVkX18kH6hnY7hTQfL4slGlLHuQiblYSjBLEpXH1r4e1NzjFxD5/k38tEn+
gAAAAABhk5IumcY2kXTSGT2Y-mJPx_iExc6XIDuRX_eHvu-B14w-KPXWam8J84Kz0OiFz6Fffd1DS1MC-
rrebeUywE5yhX4uWQ==
capturetheflag
YQHMBIQ MIDAAQOUVH LQD MZRKUVOXAI MUQ NQMUZD MZWMDI ABHX AKUQ HKATBOMOVKNMZZD VNOQNIVJQ
OXMN IDAAQOUVH KNQI, VO VI HKAAKN OK BIQ M TBYZVH/TUVJMOQ MIDAAQOUVH LQD-QFHXMNRQ
MZRKUVOXA OK QNHUDTO MNS QFHXMNRQ M IDAAQOUVH LQD, WXVHX VI OXQN BIQS YD IDAAQOUVH-LQD
HUDTOKRUMTXD OK OUMNIAVO SMOM BIVNR OXQ NKW-IXMUQS IDAAQOUVH LQD CKU M IDAAQOUVH LQD
QNHUDTOVKN MZRKUVOXA. TRT, IIX, MNS OXQ IIZ/OZI CMAVZD KC IHXQAQI BIQ OXVI TUKHQSBUQ;
OXQD MUQ OXBI HMZZQS XDYUVS HUDTOKIDIOQAI. OXQ VNVOVMZ MIDAAQOUVH HUDTOKRUMTXD-YMIQS
LQD QFHXMNRQ OK IXMUQ M IQUJQU-RQNQUMOQS IDAAQOUVH LQD CUKA OXQ IQUJQU OK HZVQNO XMI
```



```
snooping
generatingkali - /root - Unix 3.18.0.1 -
au-revoir
connecting
fallback 3936861
hint
testing
loop-itU2FsdGVkX18kH6hnY7hTQfL4slGlLHuQiblYSjBLEpXH1r4e1NzjFxD5/k38tEn+
capturetheflag
coding
find-it
get-it
VO
hi!
taking
looking
```

Figure 4. Bot replies.

Just like the first stream, most of the controller messages are unreadable while the bot replies can be read, and while they are partially different from the first test, they are similar and still don’t give an insight on what the purpose of this bot is, but it seems that one of the replies may include the current version of the OS present in the machine.



```
root@kali:~# uname -a
Linux kali 3.18.0-kali1-amd64 #1 SMP Debian 3.18.3-1~kali4 (2015-01-22) x86_64 GNU/Linux
```

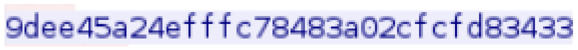
Checking on the DMZ Kali for the version, it confirms that the bot might have given information on the current version of the OS, and this may alert a security admin.

In the light of the results from this second test, it is possible to confirm the following details regarding the behaviour of this bot:

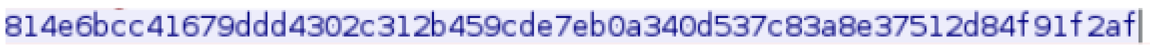
- Once executed, it begins trying to contact the controller and establish a connection starting from port 5000 onward, one at a time until a connection is established. This connection will **always** have the public Kali’s IP which in this case is `10.221.0.247`, and since it was the same address both times it is possible to assume that it is a static IP address.
- The bot will connect a second time to the open port but changing its source port. It is unclear whether this is because the first connection is a test or if it is to leave a “clean” connection as a decoy to evade controls. The only thing that is clear is that it will use the second established connection to receive commands and send replies to the controller.
- The bot will show no signs of operation on the machine beside those result on the terminal when manually executed. It runs in the background.
- The replies show that its purpose is mostly unknown, but it can definitely be used to give information that may compromise the system further.

3.3 Decrypting the messages from controller

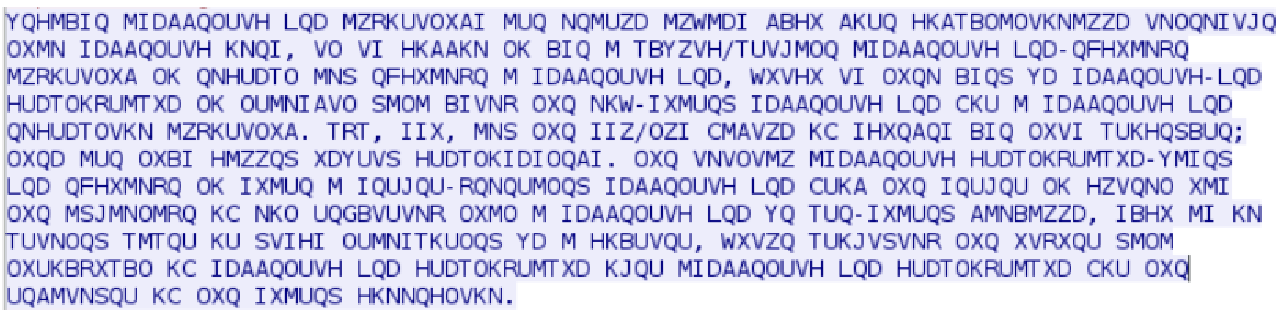
During the process of analysing the traffic, it was unclear what the commands sent from the controller were as they were all encrypted, but it was also visible that each message was encrypted using a different cipher/algorithm such as Base64, md5, sha1. For example, these encryptions are clearly different:



```
9dee45a24efffc78483a02cfcfd83433
```



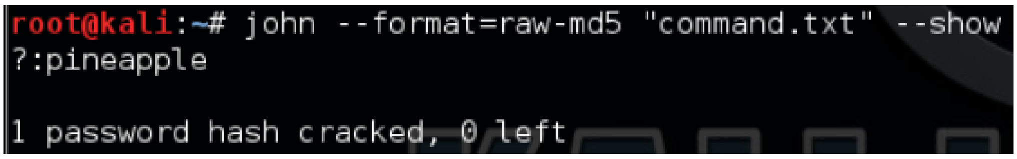
```
814e6bcc41679ddd4302c312b459cde7eb0a340d537c83a8e37512d84f91f2af
```



```
YQHMBIQ MIDAAQOUVH LQD MZRKUVOXAI MUQ NQMUZD MZWMDI ABHX AKUQ HKATBOMOVKNMZZD VNOQNIVJQ
OXMN IDAAQOUVH KNQI, VO VI HKAAKN OK BIQ M TBYZVH/TUVJMOQ MIDAAQOUVH LQD-QFHXMNRQ
MZRKUVOXA OK QNHUDTO MNS QFHXMNRQ M IDAAQOUVH LQD, WXVHX VI OXQN BIQS YD IDAAQOUVH-LQD
HUDTOKRUMTXD OK OUMNIAVO SMOM BIVNR OXQ NKW-IXMUQS IDAAQOUVH LQD CKU M IDAAQOUVH LQD
QNHUDTOVKN MZRKUVOXA. TRT, IIX, MNS OXQ IIZ/OZI CMAVZD KC IHXQAQI BIQ OXVI TUKHQSBUQ;
OXQD MUQ OXBI HMZZQS XDYUVS HUDTOKIDIOQAI. OXQ VNVOVMZ MIDAAQOUVH HUDTOKRUMTXD-YMIQS
LQD QFHXMNRQ OK IXMUQ M IQUJQU-RQNQUMOQS IDAAQOUVH LQD CUKA OXQ IQUJQU OK HZVQNO XMI
OXQ MSJMNOMRQ KC NKO UQGBVUVNR OXMO M IDAAQOUVH LQD YQ TUQ-IXMUQS AMNBMZZD, IBHX MI KN
TUVNOQS TMTQU KU SVIHI OUMNITKUOQS YD M HKBUVQU, WXVZQ TUKJVSVNR OXQ XVRXQU SMOM
OXUKBRXTBO KC IDAAQOUVH LQD HUDTOKRUMTXD KJQU MIDAAQOUVH LQD HUDTOKRUMTXD CKU OXQ
UQAMVNSQU KC OXQ IXMUQS HKNNQHOVKN.
```

Using Kali’s cipher cracking suite, it was possible to crack these messages.

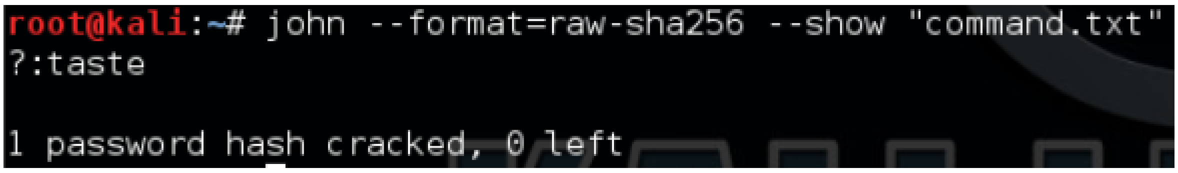
The first message is a 32-character hash, which means it was most likely encrypted using the MD5 algorithm. For this cracking test, it was decided to use the software tool “john the ripper”. The hash was saved in a *.txt* file and the cracking process gave the following result:



```
root@kali:~# john --format=raw-md5 "command.txt" --show
?:pineapple

1 password hash cracked, 0 left
```

The encoded text was “pineapple”.

The second message is a 64-character hash, so there is a high chance it was encrypted using the SHA-256 algorithm. Again, it was decided to decrypt it using “john the ripper”. The result is shown below.



```
root@kali:~# john --format=raw-sha256 --show "command.txt"
?:taste

1 password hash cracked, 0 left
```

In this case, the encrypted message from the controller was “taste”.

Focusing on the third encrypted message, the structure resembles a possible scrambled alphabet encryption.
After running a frequency test and testing out some possible occurrences, the decoded text was the following:

“because asymmetric key algorithms are nearly always much more computationally intensive than symmetric ones, it is common to use a public/private asymmetric key-exchange algorithm to encrypt and exchange a symmetric key, which is then used by symmetric-key cryptography to transmit data using the now-shared symmetric key for a symmetric key encryption algorithm. Pgp, ssh, and the ssl/tsl family of schemes use this procedure; they are thus called hybrid cryptosystems. The initial asymmetric cryptography-based key exchange to share a server-generated symmetric key from the server to client has the advantage of not requiring that a symmetric key be pre-shared manually, such as on printed paper or discs transported by a courier, while providing the higher data throughput of symmetric key cryptography over asymmetric key cryptography for the”

This highlights the possibility to decrypt the other messages from the controller using the above methods or others, where applicable, to identify the encryption algorithm and decrypt the messages.

However, the encrypted messages above do not look like a specific instruction for the bot, so, in addition to the previous list in section 3.2, it is possible to conclude that not all messages from the controller are commands to be executed but are some sort of communication feature between the two.

4. Detect and stop the attack

In this final section of the report there will be a discussion regarding possible solutions to detect and stop the communications between the bot and the controller. In the final part of this section the full configuration will be shown, and the bot will be executed once more to test the implementation of the detection and defence system.

4.1 How to defend a network

There are two main tools that allow admins to defend their networks from attacks: Intrusion Detection Systems and Firewalls.

According to Vacca (2013, p. 46), IDS detect unauthorized intrusions using one of three types of models:

- Anomaly-based systems learn the normal behaviours in a given network in order to detect any “abnormality”;
- Signature-based systems look for variations or specific signatures of suspicious network activity;
- Hybrid detection compensate for weaknesses of both the above models by combining the best of both.

Vacca also stated (p. 64) that IDS approaches can also be either host-based (the IDS monitors an individual host) or network-based (the IDS works on network packet).

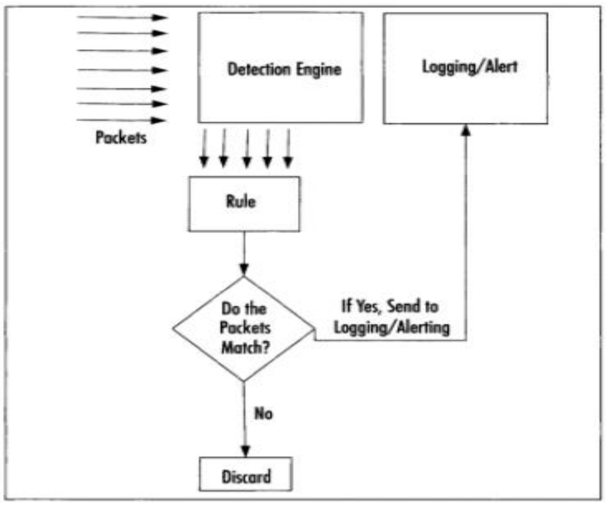
Firewalls are different from IDS because they use their set of rules to either **permit or deny** network connections. They prevent intrusions by limiting communications between networks, but they **do not** signal attacks unlike IDSs who signal suspected intrusions *after* they have taken place.
Even though firewalls are a must for anyone who wants to secure their network, they are simply not enough as there is a fundamental need to know when an attack is taking place inside the system, and that has to happen as soon as possible to allow for countermeasures.

According to Verizon’s 2021 data breach investigation report, 30% of the malware was directly installed by the hackers, 23% was sent by email and 20% was dropped from web applications. This highlights the importance of building a defence to cover these entry paths.

4.2 IDS

The IDS sensor chosen for this task is Snort.
Snort is an open-source network IDS that can perform real-time traffic analysis and packet logging on IP networks. It can be used to detect a different number of attacks: buffer overflows, port scans, etc.

Snort will be used in offline mode, reading packets from the Wireshark’s *.pcap* files that were previously shown in this report.



4.3 Snort rules definition

Snort can filter the traffic read from a tcpdump or a Wireshark *.pcap* file and filter the results using rules written in .rules files. After matching the data, it will produce an alert that can be sent to a log file that will be shown in this section.

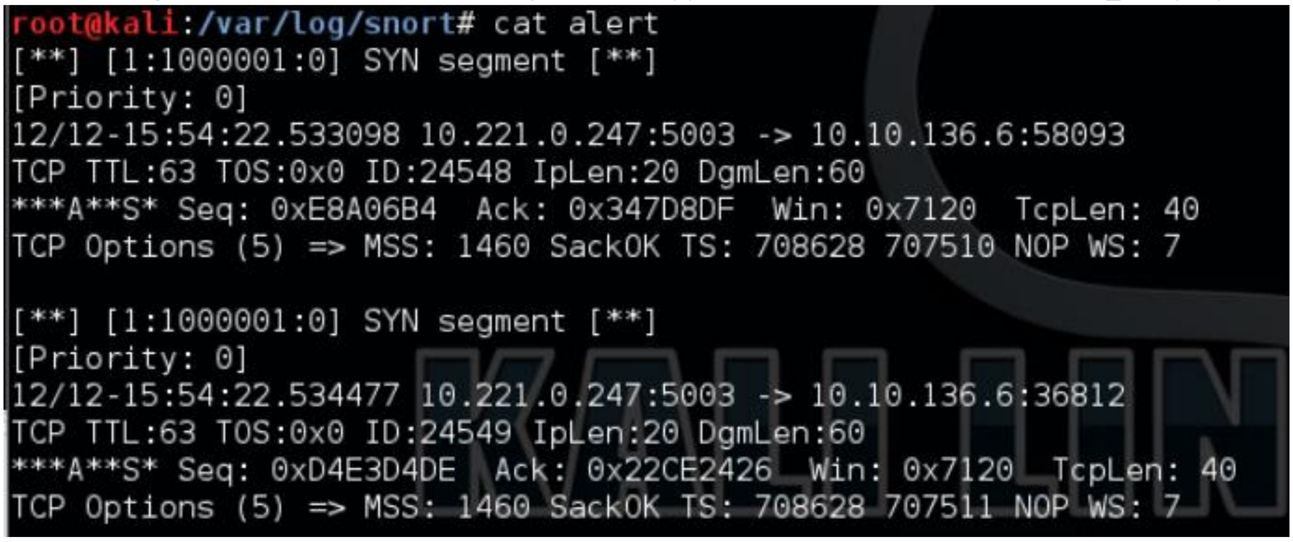
The structure of a snort rule is the following:

- Action: The action to be carried when the condition of the rule is met. In this particular case, the action will be “alert” as the aim of this implementation is to generate alerts that a security admin can read to detect an intrusion.
- Protocol: This can be TCP, UDP, or ICMP.
- Source IP: The value in the source field of the IP header. It is possible to look at all IP sources by setting this value to “any”.
- Source Port: The value in the source field of the transport protocol layer segment. It is possible to look at all ports by setting this value to “any”.
- The “->” is the direction, from source to destination.
- Destination IP: The value in the destination field of the IP header. It is possible to look at all IP sources by setting this value to “any”.
- Destination Port: The value in the destination field of the transport protocol layer segment. It is possible to look at all ports by setting this value to “any”.
- Options: The main feature of the detection engine. Gives specific information about the alert.

Example:

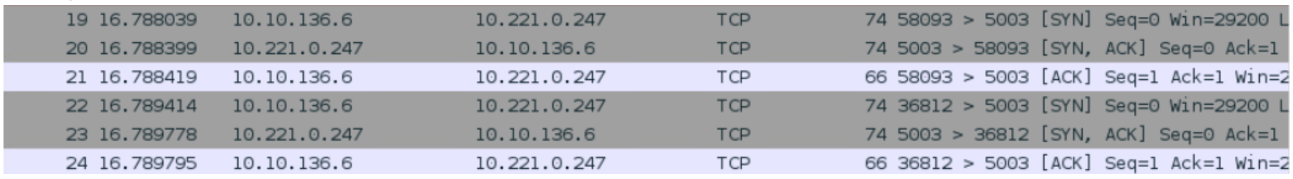
```
Alert tcp any any -> any any (flags:SA;msg:”SYN segment”;sid:1000001)
```

This rule will generate an alert when a TCP segment from any source IP/port to any destination IP/port contains a `[SYN]` flag, producing the message “SYN segment” and assigning an sid of “1000001”.
The following is the list of the alerts resulting from the application of this rule on the file first_test.pcap.



```
root@kali:/var/log/snort# cat alert
[**] [1:1000001:0] SYN segment [**]
[Priority: 0]
12/12-15:54:22.533098 10.221.0.247:5003 -> 10.10.136.6:58093
TCP TTL:63 TOS:0x0 ID:24548 IpLen:20 DgmLen:60
***A**S* Seq: 0xE8A06B4  Ack: 0x347D8DF  Win: 0x7120  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 708628 707510 NOP WS: 7

[**] [1:1000001:0] SYN segment [**]
[Priority: 0]
12/12-15:54:22.534477 10.221.0.247:5003 -> 10.10.136.6:36812
TCP TTL:63 TOS:0x0 ID:24549 IpLen:20 DgmLen:60
***A**S* Seq: 0xD4E3D4DE  Ack: 0x22CE2426  Win: 0x7120  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 708628 707511 NOP WS: 7
```

As shown during the analysis in section 3.1, there were only two successful SYN connection from the bot to the agent:



```
19 16.788039   10.10.136.6    10.221.0.247   TCP   74 58093 > 5003 [SYN] Seq=0 Win=29200 L
20 16.788399   10.221.0.247   10.10.136.6    TCP   74 5003 > 58093 [SYN, ACK] Seq=0 Ack=1
21 16.788419   10.10.136.6    10.221.0.247   TCP   66 58093 > 5003 [ACK] Seq=1 Ack=1 Win=2
22 16.789414   10.10.136.6    10.221.0.247   TCP   74 36812 > 5003 [SYN] Seq=0 Win=29200 L
23 16.789778   10.221.0.247   10.10.136.6    TCP   74 5003 > 36812 [SYN, ACK] Seq=0 Ack=1
24 16.789795   10.10.136.6    10.221.0.247   TCP   66 36812 > 5003 [ACK] Seq=1 Ack=1 Win=2
```

This proves the correctness and effectiveness of the rule.

4.4 Implementation of Snort rules

The following is a recap of the known behaviour of this botnet:

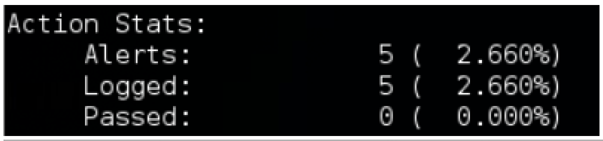
- It always tries to contact the controller on the address `10.221.0.247` and it is safe to assume it is a static IP address, otherwise the bot would not be able to contact it. And since the bot shuts down when the address is unreachable, there is no other address it is instructed to contact.
- It always tries to connect to port 5000 first, and tries the next one if unsuccessful, and then the next, and so on.
- The data is sent applying the `[PSH]` flag. When the data is larger than the maximum possible segment size and thus it is split and sent through multiple segments, only the last one will have the `[PSH]` flag set signalling the end of the message.

The first thing to look for is any attempt to establish connection from the agent to the controller. This can be achieved through the following rules:

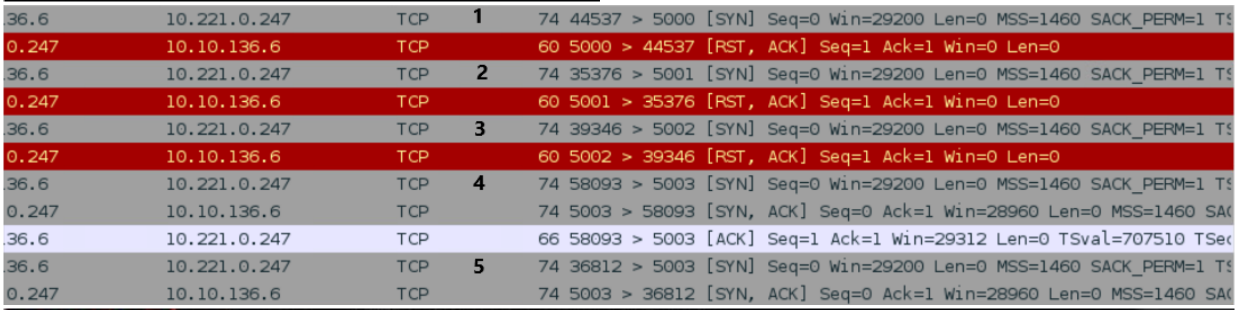
```
alert tcp 10.10.136.6 any -> 10.221.0.247 5000: (flags:S;msg:"Possible
connection attempt between agent and controller";sid:1000001;)
```

This rule will log an alert for every packet sent from the DMZ Kali's address to the public Kali's address on any port from port 5000 onward, and with a `[SYN]` flag signalling an attempt to establish a connection.

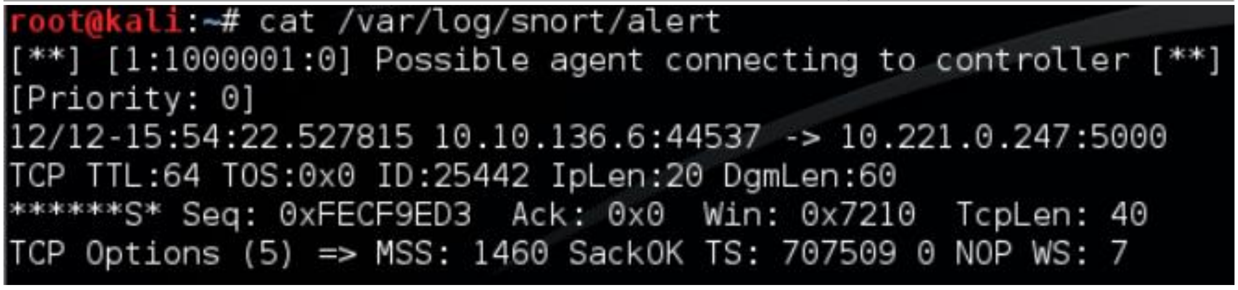
Result:



```
Action Stats:
     Alerts:              5 (  2.660%)
     Logged:              5 (  2.660%)
     Passed:              0 (  0.000%)
```



```
36.6       10.221.0.247     TCP   1   74 44537 > 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T
0.247      10.10.136.6      TCP       60 5000 > 44537 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36.6       10.221.0.247     TCP   2   74 35376 > 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T
0.247      10.10.136.6      TCP       60 5001 > 35376 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36.6       10.221.0.247     TCP   3   74 39346 > 5002 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T
0.247      10.10.136.6      TCP       60 5002 > 39346 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36.6       10.221.0.247     TCP   4   74 58093 > 5003 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T
0.247      10.10.136.6      TCP       74 5003 > 58093 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SA
36.6       10.221.0.247     TCP       66 58093 > 5003 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=707510 TSe
36.6       10.221.0.247     TCP   5   74 36812 > 5003 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T
0.247      10.10.136.6      TCP       74 5003 > 36812 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SA
```

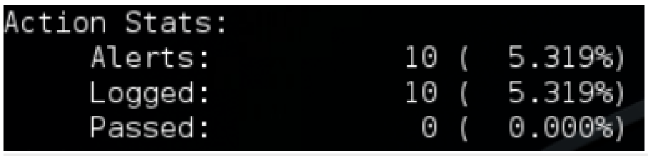


```
root@kali:~# cat /var/log/snort/alert
[**] [1:1000001:0] Possible agent connecting to controller [**]
[Priority: 0]
12/12-15:54:22.527815 10.10.136.6:44537 -> 10.221.0.247:5000
TCP TTL:64 TOS:0x0 ID:25442 IpLen:20 DgmLen:60
******S* Seq: 0xFECF9ED3  Ack: 0x0  Win: 0x7210  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 707509 0 NOP WS: 7
```

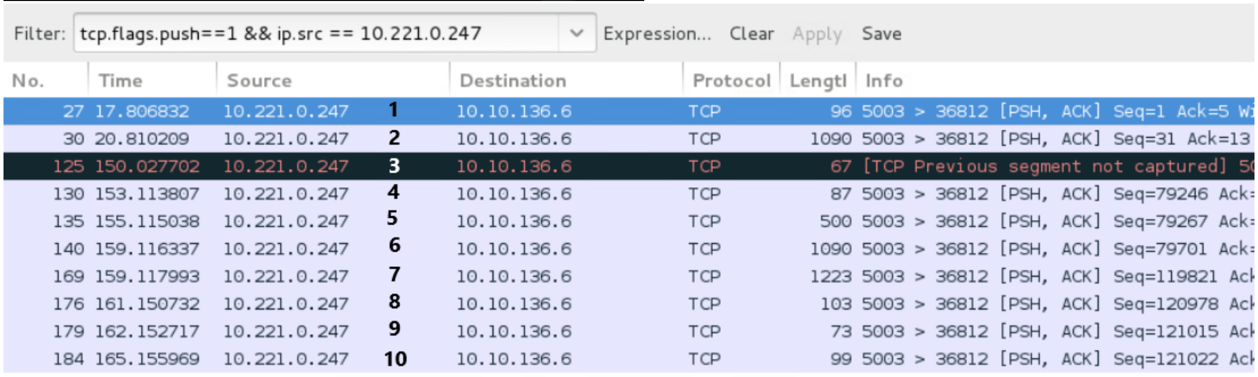
The second thing to look for is the actual communication between agent and controller. To avoid extra alerts for empty packets, a rule that only looks for `[PSH, ACK]` flags (which identifies the packets containing data, or the last packet of the flow if fragmented) is suggested. For example:

```
alert tcp 10.221.0.247 5000: -> 10.10.136.6 any (flags:PA;msg:"Message
from controller to agent";sid:1000002;)

alert tcp 10.10.136.6 any -> 10.221.0.247 5000: (flags:PA;msg:"Message
from agent to controller";sid:1000003;)
```

The first rule will signal any packet coming from the controller's machine (with the same details as the first rule) that have `[PSH, ACK]` flags meaning they contain data, and describing them with the proper message.
Result:

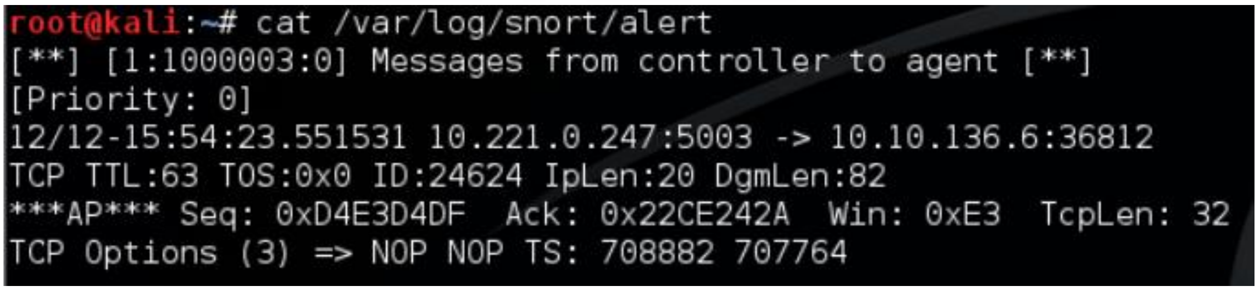


```
Action Stats:
     Alerts:           10 (  5.319%)
     Logged:           10 (  5.319%)
     Passed:            0 (  0.000%)
```



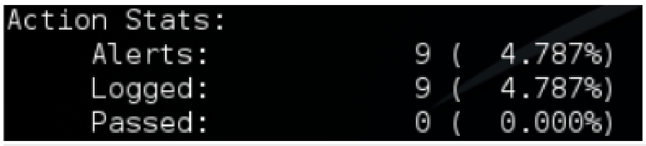
Filter: tcp.flags.push==1 && ip.src == 10.221.0.247 | Expression... | Clear | Apply | Save

No.	Time	Source		Destination	Protocol	Lengtl	Info
27	17.806832	10.221.0.247	1	10.10.136.6	TCP	96	5003 > 36812 [PSH, ACK] Seq=1 Ack=5 Wi
30	20.810209	10.221.0.247	2	10.10.136.6	TCP	1090	5003 > 36812 [PSH, ACK] Seq=31 Ack=13
125	150.027702	10.221.0.247	3	10.10.136.6	TCP	67	[TCP Previous segment not captured] 50
130	153.113807	10.221.0.247	4	10.10.136.6	TCP	87	5003 > 36812 [PSH, ACK] Seq=79246 Ack=
135	155.115038	10.221.0.247	5	10.10.136.6	TCP	500	5003 > 36812 [PSH, ACK] Seq=79267 Ack=
140	159.116337	10.221.0.247	6	10.10.136.6	TCP	1090	5003 > 36812 [PSH, ACK] Seq=79701 Ack=
169	159.117993	10.221.0.247	7	10.10.136.6	TCP	1223	5003 > 36812 [PSH, ACK] Seq=119821 Ack
176	161.150732	10.221.0.247	8	10.10.136.6	TCP	103	5003 > 36812 [PSH, ACK] Seq=120978 Ack
179	162.152717	10.221.0.247	9	10.10.136.6	TCP	73	5003 > 36812 [PSH, ACK] Seq=121015 Ack
184	165.155969	10.221.0.247	10	10.10.136.6	TCP	99	5003 > 36812 [PSH, ACK] Seq=121022 Ack

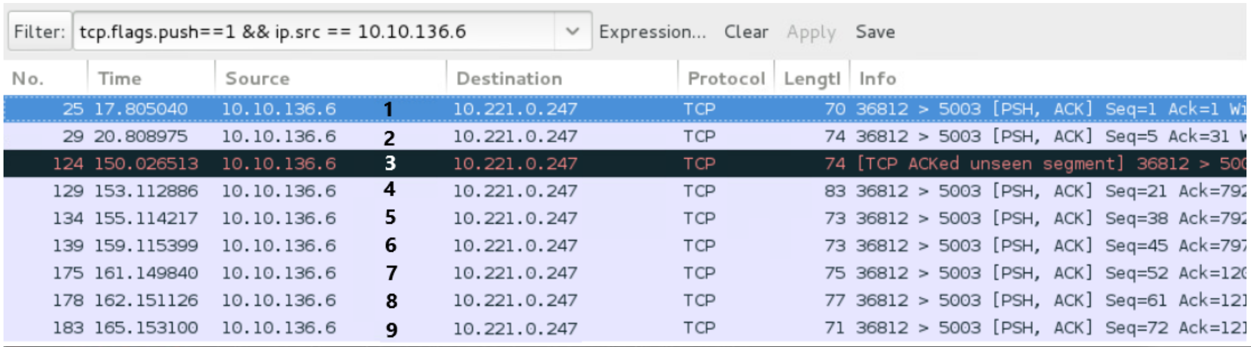


```
root@kali:~# cat /var/log/snort/alert
[**] [1:1000003:0] Messages from controller to agent [**]
[Priority: 0]
12/12-15:54:23.551531 10.221.0.247:5003 -> 10.10.136.6:36812
TCP TTL:63 TOS:0x0 ID:24624 IpLen:20 DgmLen:82
***AP*** Seq: 0xD4E3D4DF  Ack: 0x22CE242A  Win: 0xE3  TcpLen: 32
TCP Options (3) => NOP NOP TS: 708882 707764
```

The second rule will apply the same filter, but having the agent's machine as the source IP address.
Result:

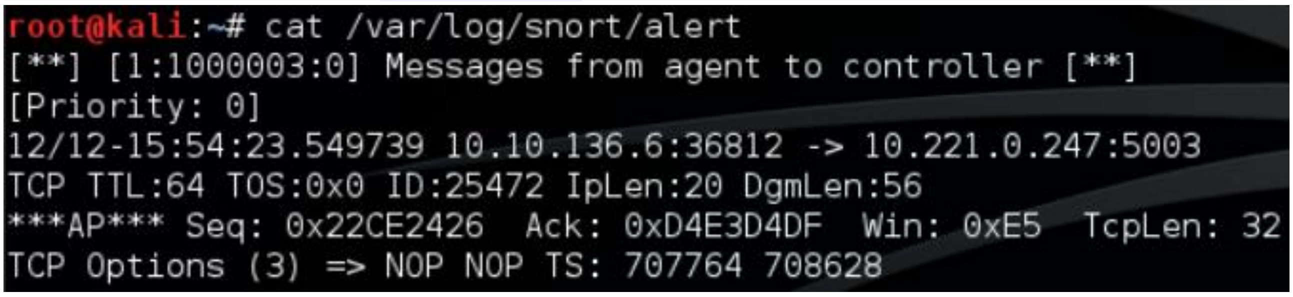


```
Action Stats:
     Alerts:            9 (  4.787%)
     Logged:            9 (  4.787%)
     Passed:            0 (  0.000%)
```



Filter: tcp.flags.push==1 && ip.src == 10.10.136.6 | Expression... | Clear | Apply | Save

No.	Time	Source		Destination	Protocol	Lengtl	Info
25	17.805040	10.10.136.6	1	10.221.0.247	TCP	70	36812 > 5003 [PSH, ACK] Seq=1 Ack=1 Wi
29	20.808975	10.10.136.6	2	10.221.0.247	TCP	74	36812 > 5003 [PSH, ACK] Seq=5 Ack=31 W
124	150.026513	10.10.136.6	3	10.221.0.247	TCP	74	[TCP ACKed unseen segment] 36812 > 500
129	153.112886	10.10.136.6	4	10.221.0.247	TCP	83	36812 > 5003 [PSH, ACK] Seq=21 Ack=792
134	155.114217	10.10.136.6	5	10.221.0.247	TCP	73	36812 > 5003 [PSH, ACK] Seq=38 Ack=792
139	159.115399	10.10.136.6	6	10.221.0.247	TCP	73	36812 > 5003 [PSH, ACK] Seq=45 Ack=797
175	161.149840	10.10.136.6	7	10.221.0.247	TCP	75	36812 > 5003 [PSH, ACK] Seq=52 Ack=120
178	162.151126	10.10.136.6	8	10.221.0.247	TCP	77	36812 > 5003 [PSH, ACK] Seq=61 Ack=121
183	165.153100	10.10.136.6	9	10.221.0.247	TCP	71	36812 > 5003 [PSH, ACK] Seq=72 Ack=121



```
root@kali:~# cat /var/log/snort/alert
[**] [1:1000003:0] Messages from agent to controller [**]
[Priority: 0]
12/12-15:54:23.549739 10.10.136.6:36812 -> 10.221.0.247:5003
TCP TTL:64 TOS:0x0 ID:25472 IpLen:20 DgmLen:56
***AP*** Seq: 0x22CE2426  Ack: 0xD4E3D4DF  Win: 0xE5  TcpLen: 32
TCP Options (3) => NOP NOP TS: 707764 708628
```

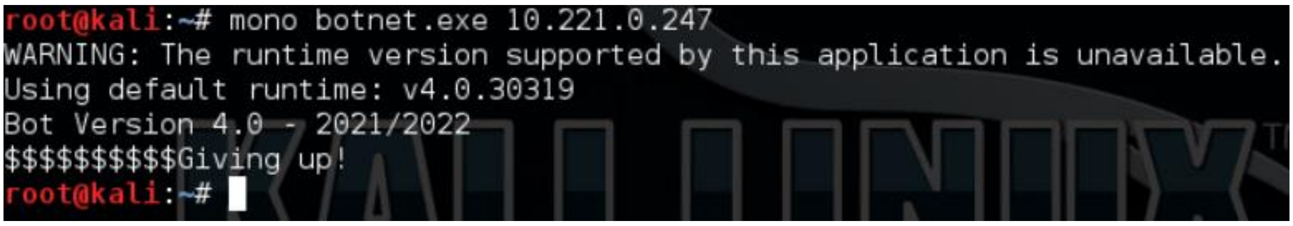
These rules together will allow a security admin to properly filter and observe the traffic avoiding as many false positives as possible. There is no data suggesting a specific content to look for in the content of the messages, nor evidence regarding other behaviours to look for.

4.5 Firewall configuration

The final step is to configure the Vyatta firewall to stop all communications between the agent and the controller.

The initial configuration uses zone-based policies, so the main intention is to keep the zones set up (no interface-to-interface firewalling) without changing their current traffic regulation.

Just like the snort rules, firewall configuration should stop traffic between the two machines when it's botnet-related. As the source port from the DMZ Kali seems to be random every time, it would be difficult to filter traffic on source port. Instead, the traffic analysis evidenced that that the agent starts communication always from port 5000 increasing by one for every failed connection, and the following screenshot of the agent failing to contact the controller shows that it gives up after exactly 10 attempts.



```
root@kali:~# mono botnet.exe 10.221.0.247
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:~#
```

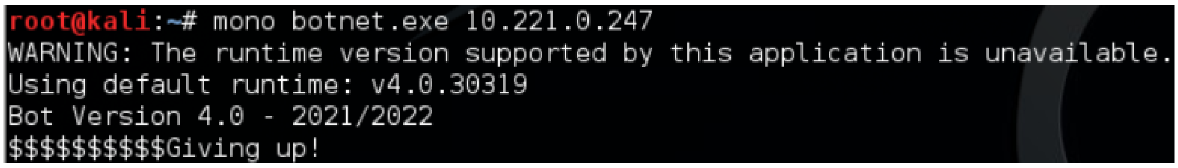
Each "$" symbol represent an attempt to connect to a port.

With all the information gained until now, the best solution is to include in the "dmz2public" firewall the option to block tcp traffic directed to `10.221.0.247` on ports 5000-5010.

```
set firewall name dmz2public rule 1 protocol tcp
set firewall name dmz2public rule 1 destination address !10.221.0.247
set firewall name dmz2public rule 1 destination port !5000-5010
```

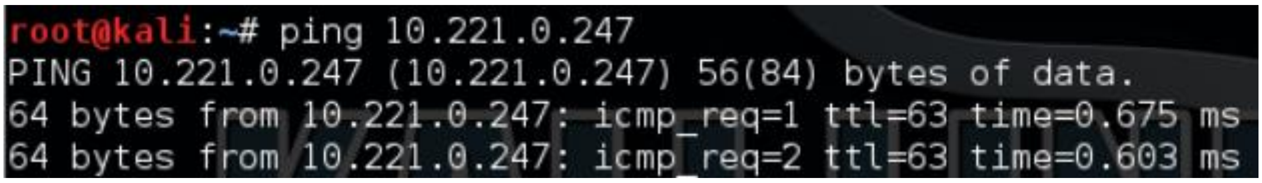
Since the previous configuration allowed all traffic, this rule only add the filter for this combination of destination address and port preventing the agent to ever contacting the controller again.
Resulting configuration:

Bot communication with controller:



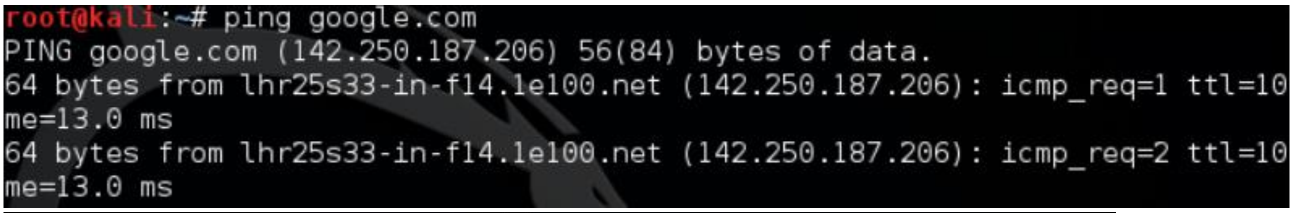
```
root@kali:~# mono botnet.exe 10.221.0.247
WARNING: The runtime version supported by this application is unavailable.
Using default runtime: v4.0.30319
Bot Version 4.0 - 2021/2022
$$$$$$$$$$Giving up!
```

Controller's machine is still reachable for valid traffic/ICMP:

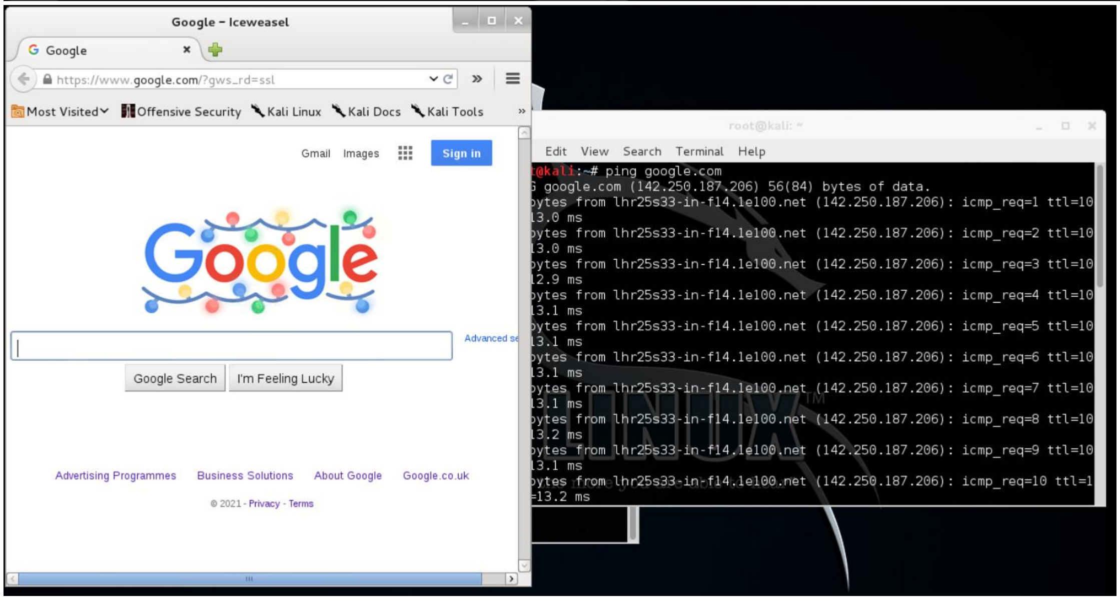


```
root@kali:~# ping 10.221.0.247
PING 10.221.0.247 (10.221.0.247) 56(84) bytes of data.
64 bytes from 10.221.0.247: icmp_req=1 ttl=63 time=0.675 ms
64 bytes from 10.221.0.247: icmp_req=2 ttl=63 time=0.603 ms
```

Agent’s machine can still connect to the internet:

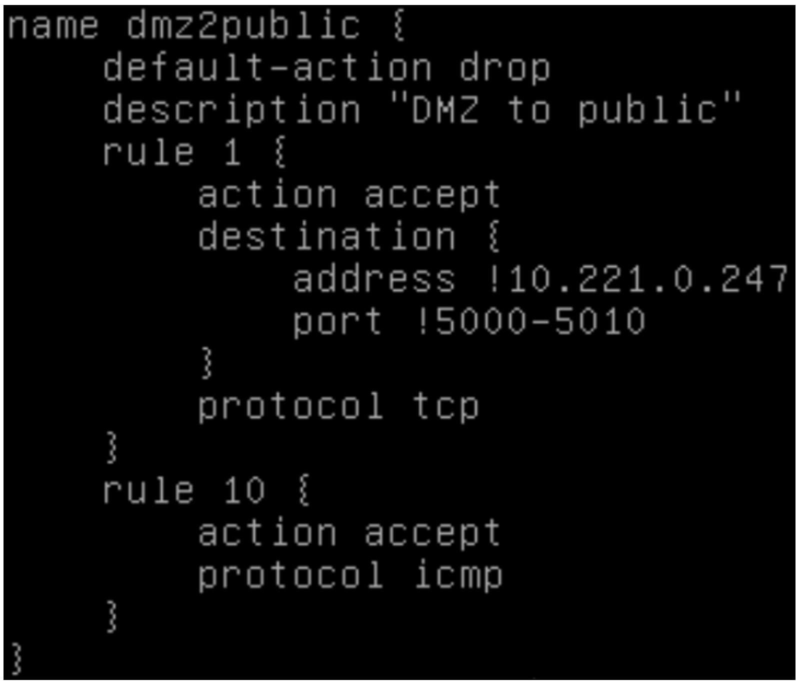


```
root@kali:~# ping google.com
PING google.com (142.250.187.206) 56(84) bytes of data.
64 bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=1 ttl=10
me=13.0 ms
64 bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=2 ttl=10
me=13.0 ms
```



```
@kali:~# ping google.com
G google.com (142.250.187.206) 56(84) bytes of data.
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=1 ttl=10
13.0 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=2 ttl=10
13.0 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=3 ttl=10
12.9 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=4 ttl=10
13.1 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=5 ttl=10
13.1 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=6 ttl=10
13.1 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=7 ttl=10
13.1 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=8 ttl=10
13.2 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=9 ttl=10
13.1 ms
bytes from lhr25s33-in-f14.1e100.net (142.250.187.206): icmp_req=10 ttl=1
=13.2 ms
```

Resulting configuration:



```
name dmz2public {
    default-action drop
    description "DMZ to public"
    rule 1 {
        action accept
        destination {
            address !10.221.0.247
            port !5000-5010
        }
        protocol tcp
    }
    rule 10 {
        action accept
        protocol icmp
    }
}
```

It is suggested to keep the *botnet.rules* file for snort as it is, in order to still be able to detect the traffic in case of a firewall failure as Vyatta seems to be error-prone.

5 References

Plohmann, D, Gerhards-Padilla, E, & Leder, F, 2011, ‘Botnets: Measurement, Detection, Disinfection and Defence’, *Enisa,* viewed 13 December 2021, <https://www.enisa.europa.eu/publications/botnets-measurement-detection-disinfection-and-defence>.

Buchanan, B, Macfarlane, R, *Lab 1: Vyatta Firewalls – Overview,* viewed 14 December 2021, <https://github.com/billbuchanan/csn09112/blob/master/week02_ids/lab/lab01_Vyatta.pdf>.

Bill Buchanan OBE, 2014, *Snort with PCap files,* Youtube, 8 January, viewed 15 December 2021, <https://www.youtube.com/watch?v=OSbZQG5AH2A&ab_channel=BillBuchananOBE>.

Vacca, J.R, 2009, *Computer and Information Security Handbook,* Morgan Kaufmann, viewed 15 December 2021, <https://books.google.it/books?id=TnE85sckwMAC&vq=IDS+network+host+signature&hl=it&source=gbs_navlinks_s>.

Buchanan, B. 2021, *Snort Analyser,* viewed 15 December 2021, <https://asecuritysite.com/forensics/snort>.

Infosec, 2021, *Basic snort rules syntax and usage,* Infosecinstitute.com, viewed 16 December 2021, <https://resources.infosecinstitute.com/topic/snort-rules-workshop-part-one/>.

VyOS, 2021, *Zone-Policy example,* docs.vyos.io, viewed 16 December 2021, <https://docs.vyos.io/en/crux/configexamples/zone-policy.html>.