Global Information Assurance Certification Paper

Practical Assignment for GIAC Intrusion Analyst Certification

Version 2.8

Alan Woodroffe

SANS Parliament Square, London, June 2001

Assignment 1 - Network Detects

1. Detect 1

Raw data taken from syslog log files generated by a FireWall, only those lines relevant to the detect are shown (the hostname of the firewall has been changed to '<firewall>').

[Reducing the font size used to display the following data will make it more readable.]

```
Dec 22 15:04:50 <firewall> PPP: Phase: bundle: Network
Dec 22 15:04 :52 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.
Dec 22 15:05:07 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.
Dec 22 15:05:3 7 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.
Dec 22 15:06:37 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.
Dec 22 15:08:36  <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.

Dec 28 22:52:23 <firewall> PPP: Phase: bundle: Network
Dec 28 22:53:38 <firewall> FILTER: Remote access filter blocks: TCP PPP
[204.71.202.119/5 050]->[62.188.156.49/1309] l=0 f=0x10.
Dec 28 22:54:54 <firewall> FILTER: Remote access filter blocks: TCP PPP
[204.71.202.119/5050] ->[62.188.156.49/1309] l=0 f=0x10.
Dec 28 22:56:11 <firewall> FILTER: Remote access filter blocks: TCP PPP
[204.71.202.119/5 050]->[62.188.156.49/1309] l=0 f=0x10.
Dec 28 22:57:27 <firewall> FILTER: Remote access filter blocks: TCP PPP
[204.71.202.119/5050] ->[62.188.156.49/1309] l=0 f=0x14.

Apr 26 13:45:45 <firewall> PPP: Phase: bundle: Network
Apr 26 13:46:09 <firewall> FILTER  : Remote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:47:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:48:09 <firewall> FILTER: Re  mote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:49:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:50:09 <firewall> FILTER: Remote   access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:51:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
```

```
Apr 26 13:52:09 <firewall> FILTER: Remote acc  ess filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x10.
Apr 26 13:53:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[205.188.8.202/5190] ->[62.188.24.76/1040] l=0 f=0x14.

Aug 16 07:39:23 <firewall> PPP: Phase: bundle: N   etwork
Aug 16 07:40:12 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0x18.
Aug 16 07:42:12 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0  x18.
Aug 16 07:44:12 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0x18.
Aug 16 07:46:12 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0x1  8.

Aug 20 18:52:07 <firewall> PPP: Phase: bundle: Network
Aug 20 18:52:46 <firewall> FILTER: Remote access filter blocks: TCP PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=56 f=0x19.
Aug 20 18:53:50 <firewall> FILTER: Remote access filter blocks: TCP    PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=56 f=0x19.
Aug 20 18:54:54 <firewall> FILTER: Remote access filter blocks: TCP PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=56 f=0x19.
Aug 20 18:55:57 <firewall> FILTER: Remote access filter blocks:    TCP PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=56 f=0x19.
Aug 20 18:57:01 <firewall> FILTER: Remote access filter blocks: TCP PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=56 f=0x19.
Aug 20 18:58:05 <firewall> FILTER: Remote access filter blo   cks: TCP PPP
[212.187.177.10/1755] ->[62.188.131.190/1114] l=0 f=0x14.
```

The above raw data was then translated by a custom written shell script on the UNIX system where the syslog logs are logged and passed into a spreadsheet to format the data and add the seconds between successive packets (where required) to produce the following:

[Reducing the font size used to display the following data will make it more readable.]

```
Date     Time     Pro Src_IP         SrcP Dst_IP         DstP Len Flags
[Notes][Secs ]

22/12/00 15:04:50
FireWall OnLine
22/12/00 15:04:52 TCP 64.124.41.195  8888 62.188.17.65   1488  21 ...AP...
8888=ddi-tcp-1
22/12/00 15:05:07 TCP 64.124.41.195  8888 62.188.17.65   1488  21 ...   AP...  15
22/12/00 15:05:37 TCP 64.124.41.195  8888 62.188.17.65   1488  21 ...AP...  30
22/12/00 15:06:37 TCP 64.124.41.195  8888 62.188.17.65   1488  21 ...AP...  60
22/12/00 15:08:36 TCP 64.124.41.195  8888 62.188.17.65   1488  21 ...AP...  119

28/12/00 22:52:23
FireWall OnLine
28/12/00 22:53:38 TCP 204.71.202.119 5050 62.188.156.49  1309   0 ...A....
5050=mmcc
28/12/00 22:54:54 TCP 204.71.202.119 5050 62.188.156.49  1309   0 ...A....  76
28/12/00 22:56:11 TCP 204.71.202.119 5050 62.188.156.49  1309   0 ...A....  77
28/12/00 22:57:27 TCP 204.71.202.119 5050 62.188.156.49  1309   0 ...A.R..  76

26/04/01 13:45:45
FireWall OnLine
26/04/01  13:46:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A....
5190=aol
26/04/01 13:47:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A....  60
26/04/01 13:48:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A....  60
26/04/01 13:49:09 TCP  205.188.8.202  5190 62.188.24.76   1040   0 ...A....  60
26/04/01 13:50:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A....  60
26/04/01 13:51:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A....  60
```

```
26/04/01 13:52:09 TCP 205.188.8.202  5190   62.188.24.76   1040   0 ...A....  60
26/04/01 13:53:09 TCP 205.188.8.202  5190 62.188.24.76   1040   0 ...A.R..  60

16/08/01 07:39:23
FireWall OnLine
16/08/01 07:40:12 TCP 64.124.41.211  8888 62.  188.143.5   1193  25 ...AP...
8888=ddi-tcp-1
16/08/01 07:42:12 TCP 64.124.41.211  8888 62.188.143.5   1193  25 ...AP...  120
16/08/01 07:44:12 TCP 64.124.41.211  8888 62.188.143.5   1193  25 ...AP...  120
16/08/01 07:46:12 TCP 64.124.41.211  8888 62.188.1   43.5   1193  25 ...AP...  120

20/08/01 18:52:07
FireWall OnLine
20/08/01 18:52:46 TCP 212.187.177.10 1755 62.188.131.190 1114  56 ...AP..F
1755=ms-streaming
20/08/01 18:53:50 TCP 212.187.177.10 1  755 62.188.131.190 1114  56 ...AP..F  64
20/08/01 18:54:54 TCP 212.187.177.10 1755 62.188.131.190 1114  56 ...AP..F  64
20/08/01 18:55:57 TCP 212.187.177.10 1755 62.188.131.190 1114  56 ...AP..F  64
20/08/01 18:57:01 TCP 212.187.177.10 1755 62.188.131.190    1114  56 ...AP..F  64
20/08/01 18:58:05 TCP 212.187.177.10 1755 62.188.131.190 1114   0 ...A.R..  64
```

1.1. Source of trace

My (small business) network.

1.2. Detect was generated by

Firewall software causing syslog entries to be generated in response to violation of firewall access filters. The firewall software provides corporate Internet access via a dial-up on demand modem. The firewall device refers to access to itself (any data destined for one of its port's IP addresses) as **'remote access'**.

IP addresses are assigned dynamically by the company's Internet Service Provider (ISP) at every dial-on-demand connection, the company in question does *not* use fixed IP addresses. Therefore the 'internal' IP addresses have not been sanitised.

The firewall performs Network Address Translation for any data being passed from the internal company networks out to the Internet and applies stateful analysis of inbound data to decide if it is 'allowed' data and to which internal address to route it.

1.2.1. Firewall syslog format

Permitted or denied access to the firewall's IP addresses is configured within the firewall's filters and any violation of these filter tables is logged to a syslog server as:

```
<Date> <Time> <Hostname> FILTER: Remote access filter blocks:
<Protocol> <Interface> [< SrcIP>/<SrcPort>] ->[<DstIP>/<DstPort>]
l=<Length> f=<Flags>.
```

For example:

```
Dec 22 15:04:52 <firewall> FILTER: Remote access filter blocks: TCP
PPP [64.124.41.195/8888] ->[62.188.17.65/1488] l=21 f=0x18.
```

Various firewall activity is logged to its syslog se rver and some of this diagnostic data is very useful when diagnosing *alleged* intrusion activity. One such type of logging is that generated when the firewall dials up its Internet link and goes online:

```
<Date> <Time> <Hostname> <Interface>: <Message>
```

For example (when going online):

```
Dec 22 15:04:50 <firewall> PPP: Phase: bundle: Network
```

The firewall refers to its dial -up interface by the name 'PPP' (for Point -to-Point Protocol). Log data was selected by various criteria, including the value of the `<Interface>` field being '`PPP`'; as this field is therefore constant it was not included in the translated, formatted data.

1.2.2. Filtered and processed data format

All date formats are shown as DD/MM/YY.

The data logged by the syslog server from the firewall is fil tered in real-time by a custom written filtering/formatting shell script which issues UNIX 'nslookup' calls to a DNS server in an attempt to identify the DNS name of any IP address which cause 'Remote access' violations. The script also simplifies some of the gathered data (e.g. IP address and ports '[1.2.3.4/5] ->[11.22.33.44/55]' into '1.2.3.4 5 11.22.33.44 55') and translates computer oriented data into human oriented data (e.g. TCP flags '0x18' into '...A.R..' meaning the TCP flags 'Ack' and 'Reset' are set).

The output of the above script is displayed on a monitor and is used to alert the company's Data Security staff (the author) to any potential intrusion.

1.3. Probability that source address was spoofed

Very low.

All the detected packets are TCP which r equire a three-way handshake for a connection to become 'established'. The observed packets (data, FIN and RST) should only be observed *after* the handshake. It *is* possible to subvert the three-way handshake but it is unlikely that these TCP sessions were subverted. It is unlikely that these are crafted packets (not needing the three -way handshake).

1.4. Description of the attack

These intercepts are examples of observing someone else's data immediately following connection to the Internet. The detection is a n example of a 'false positive' (i.e. a non-malicious event which caused an alert).

Dial-up on demand Internet access mechanisms only 'connect' to the Internet when instructed to do so. Reasons for this are typically of two types: randomly timed instance s when staff within the network served by the firewall require access to Internet resources (e.g. HTTP, SMTP); and [possibly regular]

instances when the company's email equipment checks to see if there is in-bound email for company recipients (e.g. POP3).

Internet access is provided to this company by one of the largest ISPs in the UK and their IP address re-use policy is unknown but presumed (due to the nature of these intercepts) to be that IP addresses used by one connected site are freed when that site disconnects from the Internet and are within a few minutes made available to others connecting customers.

1.5. Attack Mechanism

This is not an attack but is included to highlight scenarios observed by Internet users who do not have permanent links to the Internet.

The five above examples of this scenario show TCP packets attempting to come into to company network from various sites on the Internet and being rejected by the firewall (the firewall is configured to silently discard such packets, it does not send any response (e.g. TCP 'Reset') to the sender.

The fidelity of the data logged by the firewall does not include TCP sequence numbers so we cannot determine if the observed packets are repetitions of the same packets (TCP retransmissions) or packets containing different data. However, the times of receipt give a strong indication that these are TCP retransmissions (Stevens, p. 298-299).

When a TCP connection fails due to one side of the connection being unavailable (e.g. system crashed or network link down) the severed side (or possibly *both* sides in the case of a severed network link) start TCP retransmission where they re-send data for which they have not received and 'Acknowledgement' packet. To avoid congestion they wait for a period of time between each re-sending of a packet. After a period of time sending unsuccessful retransmissions a systems should give up and abort the connection by sending a TCP 'Reset' packet.

Some TCP implementations 'exponentially backoff' when sending re-transmissions meaning that the delay between successive retransmitted packets increases. BSD implementations double their delay up to a period of 64 seconds and then persist at 64 seconds until sending a RST after around 9 minutes. Some systems (including early Solaris systems) send their RST packet after only 2 minutes of retransmissions.

The example shown dated 22/12/00 shows the first packet being received only 2 seconds after the firewall went on-line and exponential delays from 15 to 120 seconds (shown as 119 seconds due to timing alignment differences between the sender and the firewall). The firewall log showed that the firewall stayed on-line for several minutes after these packets were logged: it is strange that no RST packet was observed.

The example shown dated 28/12/00 shows the first packet being received 15 seconds after the firewall went on-line and fixed delays of around 75 seconds. A terminating RST packet was observed.

The example shown dated 26/04/01 shows the first packet being received 24 seconds after the firewall went on-line and fixed delays of 60 seconds. A terminating RST packet was observed.

The example shown dated 16/08/01 shows the first packet being received 49 seconds after the firewall went on-line and fixed delays of 120 seconds. The firewall log showed that the firewall stayed on-line for several minutes after these packets were logged: it is strange that no RST packet was observed.

The example shown dated 20/08/01 shows the first packet being received 39 seconds after the firewall went on-line and fixed delays of 64 seconds. A terminating RST packet was observed.

The key to recognising these occurrences is that the first packet observed will be observed a period of time *less than* a retransmission time after the firewall goes on-line (e.g. the example shown from 26/04/01 with a first packet being received 24 seconds after the firewall went on-line 60 seconds between subsequent packets).

1.6. Correlation

None found.

1.7. Evidence of active targeting

This is *not* active targeting.

1.8. Severity

The severity formula is:

(criticality + lethality) - (system countermeasures + network countermeasures)

Each variable has a value from 1 (lowest) to 5 (highest).

Criticality: 5, the firewall *appears* to be the target (due to NAT).

Lethality: 1, the attack is unlikely to succeed (if it had not been blocked by the firewall the destination would have sent a TCP Reset).

System countermeasures: 5, the Operating System will send a TCP Reset.

Network countermeasures: 5, the traffic was blocked by the firewall.

(5 + 1) - (5 + 5) = -4

1.9. Defensive Recommendations

None. The 'attack' was silently blocked by the firewall.

1.10. Multiple choice test question

In a network environment where dial-on-demand Internet access is provided, which of the following is evidence of TCP data that was intended for a former user of a dynamically assigned IP address?

a. RIP data being received immediately after a dial-on-demand session starting.
b. Outbound data being sent to a DNS server immediately after a dial-on-demand session starting.
c. Repeated TCP packets being received for a not open address/port within a few seconds of a dial-on-demand session starting.
d. Multiple TCP 'Reset' packets being observed to the same address/port destination.

Answer: c

2. Detect 2

Raw data taken from syslog log files generated by FireWall and UNIX host (DNS nslookup enquiries in automated response to firewall alerts), only those lines relevant to the detect are shown.

```
Aug 16 07:44:12 <firewall> FILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0x18.
Aug 16 21:24:06 <unix_host> unix: UX:logger:INFO:64.124.41.211 DNS Name: Name:
n211.napster.com
Aug 16 07:44:31 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[158.43.254.146/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:25 <unix_host> unix: UX:logger:INFO:158.43.254.146 DNS Name: Name:
pos0-1.cr2.lnd5.gbb.uk.uu.net
Aug 16 07:44:36 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[146.188.7.234/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:30 <un ix_host> unix: UX:logger:INFO:146.188.7.234 DNS Name: Name:
so-1-0-0.TR2.LND2.Alter.Net
Aug 16 07:44:41 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[146.188.13.33/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:35 <unix_host> unix: UX:logge  r:INFO:146.188.13.33 DNS Name: Name:
so-0-0-0.IR1.DCA6.Alter.Net
Aug 16 07:44:46 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[152.63.9.210/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:40 <unix_host> unix: UX:logger:INFO:152.63.9.210 DNS    Name: Name:
0.so-0-0-0.TR1.DCA6.ALTER.NET
Aug 16 07:44:53 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[152.63.38.117/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:47 <unix_host> unix: UX:logger:INFO:152.63.38.117 DNS Name: Name:
POS6-0.BR3.DCA6.ALTER.NET
Aug 16 07:45:04 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[209.249.203.34/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:24:58 <unix_host> unix: UX:logger:INFO:209.249.203.34 DNS Name: Name:
core1-core3-oc48.iad1.above.n et
Aug 16 07:45:11 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[208.184.210.90/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:25:05 <unix_host> unix: UX:logger:INFO:208.184.210.90 DNS Name: Name:
main1colo1-core5-oc12.sjc2.above.net
Aug 16 07 :45:16 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[208.184.139.247/11] ->[62.188.143.5/11] l=32 f=0x0.
Aug 16 21:25:10 <unix_host> unix: UX:logger:INFO:208.184.139.247 DNS Name: Name:
208.184.139.247.napster.com
```

```
Aug 16 07:46:12 <firewall> F ILTER: Remote access filter blocks: TCP PPP
[64.124.41.211/8888] ->[62.188.143.5/1193] l=25 f=0x18.
Aug 16 21:26:05 <unix_host> unix: UX:logger:INFO:64.124.41.211 DNS Name: Name:
n211.napster.com
```

The above raw data was then translated by a custom writte n shell script on the UNIX system where the syslog logs are logged and passed into a spreadsheet to format the data to produce the following:

```
Date     Time     Pro  Src_IP          SrcP Dst_IP       DstP Len Flags     [Notes]
16/08/01 07:44:12 TCP  64.124  .41.211   8888 62.188.143.5 1193  25 ...AP...
16/08/01 21:24:06 DNS  64.124.41.211
n211.napster.com
16/08/01 07:44:31 ICMP 158.43.254.146  11   62.188.143.5 11    32 0x0
16/08/01 21:24:25 DNS  158.43.254.146                                    pos0  -
1.cr2.lnd5.gbb.uk.uu.net
16/08/01 07:44:36 ICMP 146.188.7.234   11   62.188.143.5 11    32 0x0
16/08/01 21:24:30 DNS  146.188.7.234                                   so    -1-0-
0.TR2.LND2.Alter.Net
16/08/01 07:44:41 I CMP 146.188.13.33   11   62.188.143.5 11    32 0x0
16/08/01 21:24:35 DNS  146.188.13.33                                   so    -0-0-
0.IR1.DCA6.Alter.Net
16/08/01 07:44:46 ICMP 152.63.9.210    11   62.188.143.5 11    32 0x0
16/08/01 21:24:40 DNS  152.63.9  .210                                    0.so  -0-
0-0.TR1.DCA6.ALTER.NET
16/08/01 07:44:53 ICMP 152.63.38.117   11   62.188.143.5 11    32 0x0
16/08/01 21:24:47 DNS  152.63.38.117                                   POS6    -
0.BR3.DCA6.ALTER.NET
16/08/01  07:45:04 ICMP 209.249.203.34  11   62.188.143.5 11    32 0x0
16/08/01 21:24:58 DNS  209.249.203.34                                  core1    -
core3-oc48.iad1.above.net
16/08/01 07:45:11 ICMP 208.184.210.90  11   62.188.143.5 11    32 0x0
16/08/01 21:25:05  DNS  208.184.210.90
main1colo1-core5-oc12.sjc2.above.net
16/08/01 07:45:16 ICMP 208.184.139.247 11   62.188.143.5 11    32 0x0
16/08/01 21:25:10 DNS  208.184.139.247
208.184.139.24 7.napster.com
16/08/01 07:46:12 TCP  64.124.41.211   8888 62.188.143.5 1193  25 ...AP...
16/08/01 21:26:05 DNS  64.124.41.211
n211.napster.com
```

2.1. Source of trace

My (small business) network.

2.2. Detect was generated by

Firewall software causing syslog entries to be generated in response to violation of firewall access filters. See Detect 1 for details.

It should be noted that the dial-on-demand firewall was online for several minutes before these packets were observed, it is *not* likely that this data was a remnant of previous use of the dynamically assigned IP address.

2.2.1. Firewall syslog format

Automated DNS lookups performed by the UNIX host where the syslog logs are logged appear formatted as:

```
<Date> <Time> <Hostname> unix : UX:logger:INFO:<IPAddr> DNS Name: Name:
<DNSName>
```

For example:

```
Aug 16 21:20:06 <unix_host> unix: UX:logger:INFO:64.124.41.211 DNS
Name: Name:    n211.napster.com
```

Other lines are formatted as detailed in Detect 1.

It should be noted that there are **timing differences** between the times logged to the syslog server by the firewall and those logged by the UNIX system holding the syslog logs. The dates and time logged by a syslog server are those supplied by the *logging* equipment, not the syslog server. The firewall being used has a unreliable real-time clock and it is apparent that sometimes the firewall's clock was not correctly set (as on this occasion). **This anomaly highlights the importance of ensuring that clocks and timing generators used in all equipment which will be collated should be consistent.**

2.2.2. Filtered and processed data format

The data logged by the syslog server from the firewall is filtered in real-time as detailed in Detect 1. Lines shown here for ICMP packets (not shown in Detect 1 above) include a flag field which shows the ICMP message code (in hexadecimal) contained within the packet.

2.3. Probability that source address was spoofed

Medium.

ICMP message type 11 message code 0 packets are used to indicate 'Time To Live exceeded in transit' (http://www.iana.org/assignments/icmp-parameters) and they are used as a one-way error indication when the TTL counter in an IP packet decrements to zero on route to its destination *before* reaching that destination. It is unlikely that these packets would have been crafted with spoof source addresses because, other than consuming a small amount of network bandwidth and processor time, they would have no detrimental effect on the recipient.

2.4. Description of the attack

This intercept is the result of a traceroute command being issued from an internal (on the protected side of the firewall) Windows system. It appears to be an attack on the Internet interface of the firewall but it can be shown that this appearance is a result of the NAT mechanism used within the firewall (see 'attack mechanism' below).

The Windows traceroute command (named 'tracert') works by sending ICMP 'echo request' packets towards the host being traced. The command first sends three packets with their TTL set to 1 which will cause the first recipient (possibly a router) to decrement the TTL to 0 and then, if it is *not* the destination, report that it cannot forward the packet because the TTL is 0. The reporting back is done by sending an ICMP 'Time To Live exceeded in transit' from the intermediate router to the tracert host thus identifying the IP address of the intermediate router. Subsequent packets are sent by 'tracert' with

increasing TTL values so that routers nearer to the destination are identified until the destination itself is reached.

The firewall is configured to reject packets directed at it from IP addresses that are not in the firewall's table of active connections (the firewall is 'stateful'). Therefore if an ICMP 'Time exceeded' message is received from an IP address to which no outbound packet has been sent then the firewall will reject the packet thinking it to be unsolicited.

2.5. Attack Mechanism

This is not an attack but is included to provide an example of a scenario that may be *perceived* to be an attack. **If recognised, an 'attack' of this type can quickly be disregarded, leaving analysts to concentrate on other more important scenarios.**

Log analysis was carried out to find if any 'tracert' had been logged immediately prior to the 'attack'. Selected lines from the syslog are shown:

```
Aug 16 07:44:31 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0.
Aug 16 07:44:31 <firewall> NAT: Open ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 07:44:31 <firewall> NAT: Close ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3] Pkts 3 0, Bytes 276 0.
Aug 16 07:44:31 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[158.43.254.146/11] ->[62.188.1 43.5/11] l=32 f=0x0.

Aug 16 07:44:34 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0.
Aug 16 07:44:34 <firewall> NAT: Open ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 07:44:35 <firewa ll> NAT: Close ICMP [<Win_Host>/3]  ->[62.188.143.5/3] -
>[64.124.41.211/3] Pkts 3 0, Bytes 276 0.

Aug 16 07:44:36 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0.
Aug 16 07:44:36 <firewall> NAT: Open ICMP [<  Win_Host>/3] ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 07:44:36 <firewall> NAT: Close ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3] Pkts 3 0, Bytes 276 0.
Aug 16 07:44:36 <firewall> FILTER: Remote access filter blocks: ICMP PPP
[146.188.7.234 /11]->[62.188.143.5/11] l=32 f=0x0.

Aug 16 07:44:39 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0.
Aug 16 07:44:39 <firewall> NAT: Open ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 0 7:44:40 <firewall> NAT: Close ICMP [<Win_Host>/3]  ->[62.188.143.5/3] -
>[64.124.41.211/3] Pkts 3 0, Bytes 276 0.

<similar lines removed for brevity>

Aug 16 07:45:15 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68  f=0x0.
Aug 16 07:45:15 <firewall> NAT: Open ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 07:45:16 <firewall> NAT: Close ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3] Pkts 3 0, Bytes 276 0.
Aug 16 07:45:16 <firewall> FILTER:  Remote access filter blocks: ICMP PPP
[208.184.139.247/11] ->[62.188.143.5/11] l=32 f=0x0.
```

```
Aug 16 07:45:19 <firewall> FILTER: Outbound filter accepts : ICMP ep1
[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0.
Aug 16 07:45:19 <firewall> NAT: Open ICMP [<Win_H  ost>/3] ->[62.188.143.5/3] -
>[64.124.41.211/3].
Aug 16 07:45:36 <firewall> NAT: Close ICMP [<Win_Host>/3]   ->[62.188.143.5/3] -
>[64.124.41.211/3]  Pkts 3 3, Bytes 276 276.
```

Note[1]: References to the IP address of the issuing Windows host systems have been changed to <Win_Host>.

Questions arise from analysis of this information:

a) Why does the firewall report '`Open ICMP [<Win_Host>/3] ->[62.188.143.5/3] ->[64.124.41.211/3]`'? ICMP message type 3 is 'Destination unreachable' which is not appropriate in this scenario. The <Win_Host> system asked for '`[<Win_Host>/8] ->[64.124.41.211/8] l=68 f=0x0`' which is ICMP message type 8, message code 0 (echo request).

b) Why does the second group of 3 ICMP packets (sent around 07:44:34 - 07:44:35) *not* show a corresponding 'time excee ded' response?

Further analysis was carried out to establish the characteristics of tracert's operation in a controlled environment so that comparisons could be made to this observation 'from the wild'. A tracert command was issued from another Windows system, through a router to the same firewall to a target [Solaris] system behind the firewall. Microsoft's 'Network Monitor' was used to monitor packets in/out of the issuing Windows system; tcpdump was used on both the router and the target system; syslo g entries (as above) were collected from the firewall. The resultant intelligence is shown (some columns have been removed to save page width in the interests of readability):

```
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>

sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<router>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP       Time Exceeded while trying to deliver to <target>
<router>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <ta    rget>
<router>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewall_inside>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewall_inside>  <sender>
sender:(NetMon)   ICMP      Echo,     From   <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewall_inside>  <sender>
```

```
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Echo Reply, To <sender> From <target>
<target>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Echo Re  ply, To <sender> From <target>
<target>  <sender>
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
sender:(NetMon)   ICMP      Echo Reply, To <sender> From <target>
<target>  <sender>

router:(tcpdump)  12:23:45.39 <sender> > <target>: icmp: echo request [ttl 1] (id
44299)
router:(tcpdump)  12:23:45.39 <sender> > <target>: icmp: echo request [ttl 1] (id
44555)
router:(tcpdump)  12:23:45.40 <sender> > <target>: icmp: echo req   uest [ttl 1] (id
44811)
router:(tcpdump)  12:23:46.39 <sender> > <target>: icmp: echo request (ttl 2, id
45067)
router:(tcpdump)  12:23:46.39 <sender> > <target>: icmp: echo request (ttl 2, id
45323)
router:(tcpdump)  12:23:46.39 <sender> > <target>: icmp:    echo request (ttl 2, id
45579)
router:(tcpdump)  12:23:47.39 <sender> > <target>: icmp: echo request (ttl 3, id
45835)
router:(tcpdump)  12:23:47.40 <sender> > <target>: icmp: echo request (ttl 3, id
46091)
router:(tcpdump)  12:23:47.40 <sender> > <target  >: icmp: echo request (ttl 3, id
46347)

firewall:(syslog) Aug 31 12:23:49 <firewall> FILTER: Outbound filter accepts : ICMP
ep0 [<sender>/8] ->[<target>/8] l=68 f=0x0.
firewall:(syslog) Aug 31 12:23:49 <firewall> NAT: Open ICMP [<sender>/1]    -
>[<firewall_dmz >/1]->[<target>/1].
firewall:(syslog) Aug 31 12:24:05 <firewall> NAT: Close ICMP [<sender>/1]    -
>[<firewall_dmz>/1] ->[<target>/1] Pkts 3 3, Bytes 276 276.

target:(tcpdump)  12:23:48.428992 <firewall_dmz> > <target>: icmp: echo request
[ttl 1]
target:(tcpdum p)  12:23:48.429302 <target> > <firewall_dmz>: icmp: echo reply (DF)
target:(tcpdump)  12:23:48.432430 <firewall_dmz> > <target>: icmp: echo request
[ttl 1]
target:(tcpdump)  12:23:48.432510 <target> > <firewall_dmz>: icmp: echo reply (DF)
target:(tcpdump)   12:23:48.436247 <firewall_dmz> > <target>: icmp: echo request
[ttl 1]
target:(tcpdump)  12:23:48.436317 <target> > <firewall_dmz>: icmp: echo reply (DF)
```

These logs were then placed in order of occurrence so that the flow of data may be observed, thus:

```
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:45.39 <sender> > <target>: icmp: echo request [ttl 1] (id
44299)
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver    to <target>
<router>  <sender>

sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:45.39 <sender> > <target>: icmp: echo request [ttl 1] (id
44555)
sender:(NetMon)   ICMP       Time Exceeded while trying to deliver to <target>
<router>  <sender>
```

```
sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:45.40 <sender> > <target>: icmp: echo request [    ttl 1] (id
44811)
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<router>  <sender>


sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:46.3  9 <sender> > <target>: icmp: echo request (ttl 2, id
45067)
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewall_inside>  <sender>

sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:46.39 <sender> > <target>: icmp: echo request (ttl 2, id
45323)
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewall_inside>  <sender>

sender:(NetMon)   ICMP      E  cho,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:46.39 <sender> > <target>: icmp: echo request (ttl 2, id
45579)
sender:(NetMon)   ICMP      Time Exceeded while trying to deliver to <target>
<firewal l_inside>  <sender>


sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:47.39 <sender> > <target>: icmp: echo request (ttl 3, id
45835)
firewall:(syslog) Aug 31 12:23:49 <firewall>   FILTER: Outbound filter accepts : ICMP
ep0 [<sender>/8] ->[<target>/8] l=68 f=0x0.
firewall:(syslog) Aug 31 12:23:49 <firewall> NAT: Open ICMP [<sender>/1]    -
>[<firewall_dmz>/1] ->[<target>/1].
target:(tcpdump)  12:23:48.428992 <firewall_dmz> > <target>: icmp:    echo request
[ttl 1]
target:(tcpdump)  12:23:48.429302 <target> > <firewall_dmz>: icmp: echo reply (DF)
sender:(NetMon)   ICMP      Echo Reply, To <sender> From <target>
<target>  <sender>

sender:(NetMon)   ICMP      Echo,     From <sende   r> To   <target>
<sender>  <target>
router:(tcpdump)  12:23:47.40 <sender> > <target>: icmp: echo request (ttl 3, id
46091)
target:(tcpdump)  12:23:48.432430 <firewall_dmz> > <target>: icmp: echo request
[ttl 1]
target:(tcpdump)  12:23:48.4  32510 <target> > <firewall_dmz>: icmp: echo reply (DF)
sender:(NetMon)   ICMP      Echo Reply, To <sender> From <target>
<target>  <sender>

sender:(NetMon)   ICMP      Echo,     From <sender> To   <target>
<sender>  <target >
router:(tcpdump)  12:23:47.40 <sender> > <target>: icmp: echo request (ttl 3, id
46347)
target:(tcpdump)  12:23:48.436247 <firewall_dmz> > <target>: icmp: echo request
[ttl 1]
target:(tcpdump)  12:23:48.436317 <target> > <firewall_dmz>: icmp: echo reply     (DF)
sender:(NetMon)   ICMP      Echo Reply, To <sender> From <target>
<target>  <sender>


firewall:(syslog) Aug 31 12:24:05 <firewall> NAT: Close ICMP [<sender>/1]    -
>[<firewall_dmz>/1] ->[<target>/1] Pkts 3 3, Bytes 276 276.
```

Note[1]: The Windows commands 'tracert -d -h 1 <target>', 'tracert -d -h 2 <target>' and 'tracert -d -h 3 <target>' (with incrementing '-h' maximum hop count arguments) were used to verify that issuing ICMP echo requests with a hop count up to and including the firewall do not cause the firewall to open an ICMP path to the target. Only the final tracert command (with '-h 3' to use a maximum hop count of 3) caused the 'ICMP ep0 [<sender>/8] ->[<target>/8] l=68 f=0x0' syslog entry from the firewall.

Note[2]: The router is a UNIX host (UnixWare), tcpdump (Version 3.0.4) running on one of its interfaces does not log packets *transmitted* by that interface, only those received by it.

Note[3]: The target system is also a UNIX host (Solaris), tcpdump (Version 3.6) running on one of its interfaces logs packets both *transmitted and received* by that interface.

Note[4]: The firewall allows ICMP packets to reach <target> and records the number of packets sent/received and total number of bytes sent/received (see last line in the above traces). The firewall records that 3 packets were sent [by the firewall, to <target>] and that 3 were received, 'Pkts 3 3' respectively. The firewall records that 276 bytes were sent in total in these packets and that 276 bytes were received, 'Bytes 276 276' respectively. It we assume that each packets was the same size then dividing 276 by 3 gives us 92 bytes per packet. Using our knowledge of IP and ICMP headers we can decompose the byte count into: IP header (20 bytes), ICMP header (4 bytes) leaving 68 bytes for ICMP data content. Checking the firewall log entries we can see that the firewall does indeed log the data length as 68 (' l=68').

The questions from above were then re-visited:

a) This time the firewall reports 'Open ICMP [<sender>/1] ->[<firewall_dmz>/1] ->[<target>/1]' when the Win_Host system requested '[<sender>/8] ->[<target>/8] l=68 f=0x0'. Could it be that the firewall is *incorrectly* reporting the outbound ICMP message type? Bear in mind that TCP and UDP packets contain their port number in the 16 bits at byte offsets 2 and 3 in TCP and UDP packets and that an ICMP packet's message type is an 8 bit value located at byte offset 0 in the ICMP packet. Offset 2 and 3 in ICMP packets contain a 16-bit checksum; could it be that this is being incorrectly interpreted as the message type? This could account for the fact that it is recorded differently on different occasions.

b) In this example we either see a 'time exceeded' response or the final 'echo reply' for *all* sent packets. An explanation for the lack of responses in the above 'real world' example could be that the items of equipment that should have replied were perhaps configured *not* to send ICMP error packets.

2.6. Correlation

Searching yahoo.com for 'ICMP 11' revealed a report at //false.net/ipfilter/2001_03/0285.html which discusses similar anomalies.

2.7. Evidence of active targeting

This is *not* active targeting.

2.8. Severity

The severity formula exp lained in Detect 1 above.

Criticality: 5, the firewall *appears* to be the target (due to NAT).

Lethality: 3, this *may* be a DoS attack on the firewall (a *recognised* DoS attack would be a 4).

System countermeasures: 3, the Operating System should cope with the data.

Network countermeasures: 5, the traffic was blocked by the firewall.

(5 + 3) - (3 + 5) = 0

2.9. Defensive Recommendations

None. The 'attack' was silently blocked by the firewall. It may be prudent in this type of environment (i.e. using a firewal l with Network Address Translation) to modify firewall rules to silently discard inbound ICMP 'time exceeded in transit' packets as the firewall may not be able to determine to which 'inside' IP address it should forward them.

2.10. Multiple choice test question

When the Windows 'tracert' command is used, why might some responses appear to be missing?

a. some ICMP packets are often routed to the wrong destination.
b. intermediate routers may be configured *not* to send ICMP error packets.
c. firewalls never pass o n ICMP error packets.
d. some systems respond to 'tracert' with packets using unrecognisable protocols.

Answer: b

3. Detect 3

Raw data taken from syslog log files generated by FireWall, only those lines relevant to the detect are shown.

```
Apr 17 14:27:35 <f irewall> FILTER: Remote access filter blocks: UDP PPP
[62.188.26.244/1028] ->[62.188.26.92/5632] l=2.
Apr 17 14:27:35 <firewall> FILTER: Remote access filter blocks: UDP PPP
```

```
[62.188.26.244/1028] ->[62.188.26.92/22] l=2.
Apr 17 14:29:03 <firewall> FILTER: Rem  ote access filter blocks: UDP PPP
[62.188.26.244/1029] ->[62.188.26.92/5632] l=2.
Apr 17 14:29:03 <firewall> FILTER: Remote access filter blocks: UDP PPP
[62.188.26.244/1029] ->[62.188.26.92/22] l=2.
Apr 17 14:30:07 <firewall> FILTER: Remote access filter bl   ocks: UDP PPP
[62.188.26.244/1030] ->[62.188.26.92/5632] l=2.
Apr 17 14:30:07 <firewall> FILTER: Remote access filter blocks: UDP PPP
[62.188.26.244/1030] ->[62.188.26.92/22] l=2.
```

The above raw data was then translated by a custom written shell script on th e UNIX system where the syslog logs are logged and passed into a spreadsheet to format the data to produce the following:

```
Date     Time     Pro Src_IP        SrcP Dst_IP       DstP Len Secs
17/04/01 14:27:35 UDP 62.188.26.244 1028 62.188.26.92 5632   2
17/04/01 14:27:35 UDP 62.188.26.244 1028 62.188.26.92 22     2 0:00:00

17/04/01 14:29:03 UDP 62.188.26.244 1029 62.188.26.92 5632   2 0:01:28
17/04/01 14:29:03 UDP 62.188.26.244 1029 62.188.26.92 22     2 0:00:00

17/04/01 14:30:07 UDP 62.188.26.244 1030 62  .188.26.92 5632   2 0:01:04
17/04/01 14:30:07 UDP 62.188.26.244 1030 62.188.26.92 22     2 0:00:00
```

3.1. Source of trace

My (small business) network.

3.2. Detect was generated by

Firewall software causing syslog entries to be generated in response to violation of firewall access filters. See Detects above for details.

It should be noted that the dial-on-demand firewall was online for several minutes before these packets were observed, it is *not* likely that this data was a remnant of previous use of the dynamically a ssigned IP address.

3.2.1. Firewall syslog format

The lines seen above represent UDP packets, the format of the lines is similar to that shown for TCP packets in Detect 1 except that UDP packets do not show any 'Flag' data (they do not have 'flags' as TCP packets do nor 'message codes' as ICMP packets do).

3.2.2. Filtered and processed data format

As for the Detects above.

3.3. Probability that source address was spoofed

Medium.

UDP packets do *not* require a three-way handshake to establish them (as TCP packets do) and it is possible to send spoofed source address UDP packets fairly easily.

Why would anyone send just 6 UDP packets to my equipment? It does not seem to be a DoS attempt. The answer is probably that this is a reconnaissance attempt. There is no point in spoo fing source addresses in

reconnaissance attempts because you will not learn the result of the reconnaissance. Some reconnaissance processes *do* use spoof addresses (as well as their genuine address) to confuse attacked systems by sending packets from many different source addresses hoping that the systems will respond to all the packets and that an analyst will not know which of the source addresses is the genuine address.

3.4. Description of the attack

This intercept appears to be a reconnaissance attack.
See CVE-2000-0273 and Bugtraq BID:1095 for details.
It is known by the name 'PCanywhere Denial of Service Vulnerability' and is attributed to Frankie Zie <root@cnns.net> on 9 Apr 2000.

3.5. Attack Mechanism

Information at www.networkice.com/advice/Exploits/Ports/groups/PCanywhere/default.htm provides further information about the mechanism used in this vulnerability.

Port 5632 is used by PCanywhere software to 'ping' a syste m to which a connection is being attempted to see if the PCanywhere service is available on the target system. A programming bug swapped the bytes in order versions (the 16 bit number 5632 when byte swapped is 22).

The networkice reference also states 'I f the user doesn't know the IP address, PCanywhere will ping the entire local address range with these packets looking for servers. These scans are frequently seen by home users from their neighbors.' Looking at the IP addresses in use in this instance we see source 62.188.26.244 and destination 62.188.26.92 so we seem to have an example of this.

3.6. Correlation

A person identified as 'binette@home' discussed a pattern very similar to the above on 8 January 2001 and the item is included in Matt Fearnow's SANS handler's diary at www.sans.org/y2k/010801.htm .

3.7. Evidence of active targeting

This is *probably not* active targeting. The source IP address is within the same Class C range as the dynamically assigned firewa ll address in use at the time. It is probable that the source address is scanning addresses within a particular range of addresses to see which respond. If any systems respond it is presumed that the source address will then conduct more detailed reconnaissance or initiate an exploit on the recognised architecture/service.

3.8. Severity

The severity formula explained in Detect 1 above.

Criticality: 5, the firewall *appears* to be the target (due to NAT).

Lethality: 2, this appears to be reconnaissance.

System countermeasures: 4, the Operating System should reply with 'port Unreachable', as the systems do not listen on UDP ports 22 or 5632.

Network countermeasures: 5, the traffic was blocked by the firewall.

(5 + 2) - (4 + 5) = -2

3.9. Defensive Recommendations

None. The 'attack' was silently blocked by the firewall.

3.10. Multiple choice test question

Why are packets sent from custom written programs sometimes observed communicating with unexpected destination port numbers?

a. Because programmers sometimes forget to u se 'network byte ordered' data.
b. Because the programs often use the wrong protocol.
c. Because programs cannot guarantee which port they will use.
d. Because of hardware faults with network equipment.

Answer: a

4. Detect 4

Raw data taken from syslog log files generated by FireWall, only those lines relevant to the detect are shown.

```
Dec 11 16:14:25 <firewall> FILTER: Remote access filter blocks: UDP PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=268.
Dec 11 16:14:40 <firewall> FILTER: Remote access filte   r blocks: UDP PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=268.
Dec 11 16:14:55 <firewall> FILTER: Remote access filter blocks: UDP PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=268.
Dec 11 16:14:55 <firewall> FILTER: Remote access filter blocks: UDP    PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=268.
Dec 11 16:15:01 <firewall> FILTER: Remote access filter blocks: UDP PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=68.
Dec 11 16:15:10 <firewall> FILTER: Remote access filter blocks: UDP PPP
[194.130.10 2.189/500] ->[62.188.19.92/500] l=268.
Dec 11 16:15:10 <firewall> FILTER: Remote access filter blocks: UDP PPP
[194.130.102.189/500] ->[62.188.19.92/500] l=268.

Dec 12 09:36:28 <firewall> FILTER: Remote access filter blocks: UDP PPP
[193.131.113.187/500] ->[62.188.22.95/500] l=84.

Apr  6 21:48:13 <firewall> FILTER: Remote access filter blocks: UDP PPP
[206.9.231.3/500] ->[62.188.130.16/500] l=80.
Apr  6 21:48:28 <firewall> FILTER: Remote access filter blocks: UDP PPP
[206.9.231.3/500] ->[62.188.130.16/500] l=8  0.
Apr  6 21:48:43 <firewall> FILTER: Remote access filter blocks: UDP PPP
[206.9.231.3/500] ->[62.188.130.16/500] l=80.
```

The above raw data was then translated by a custom written shell script on the UNIX system where the syslog logs are logged and passed into a spreadsheet to format the data to produce the following:

```
Date     Time     Pro Src_IP          SrcP Dst_IP        DstP Len Secs
11/12/00 16:14:25 UDP 194.130.102.189  500 62.188.19.92   500 268
11/12/00 16:14:40 UDP 194.130.102.189  500 62.188.19.9   2   500 268 0:00:15
11/12/00 16:14:55 UDP 194.130.102.189  500 62.188.19.92   500 268 0:00:15
11/12/00 16:14:55 UDP 194.130.102.189  500 62.188.19.92   500 268 0:00:00
11/12/00 16:15:01 UDP 194.130.102.189  500 62.188.19.92   500 268 0:00:06
11/12/00 16:15 :10 UDP 194.130.102.189  500 62.188.19.92   500 268 0:00:09
11/12/00 16:15:10 UDP 194.130.102.189  500 62.188.19.92   500 268 0:00:00

12/12/00 09:36:28 UDP 193.131.113.187  500 62.188.22.95   500  84

06/04/01 21:48:13 UDP 206.9.231.3      500 62.188.130.   16  500  80
06/04/01 21:48:28 UDP 206.9.231.3      500 62.188.130.16  500  80 0:00:15
06/04/01 21:48:43 UDP 206.9.231.3      500 62.188.130.16  500  80 0:00:15
```

4.1. Source of trace

My (small business) network.

4.2. Detect was generated by

Firewall software causing syslog entries to be generated in response to violation of firewall access filters. See Detects above for details.

It should be noted that the dial-on-demand firewall was online for several minutes before these packets were observed, it is *not* likely that this data was a remnant of previous use of the dynamically assigned IP address.

4.2.1. Firewall syslog format

As for the Detects above.

4.2.2. Filtered and processed data format

As for the Detects above.

4.3. Probability that source address was spoofed

Medium.

UDP packets do *not* require a three-way handshake to establish them (as TCP packets do) and it is possible to send spoofed source address UDP packets fairly easily.

As in Detect 3, this is probably a reconnaissance attempt. There is no point in spoofing source addresses in reconnaissance attempts because you will not learn the result of the reconnaissance.

4.4. Description of the attack

UDP port 500, IANA port name 'isakmp' (Internet Security Association and Key Management Protocol) is used for encryption key exchange when establishing secure sessions for various reasons, including VPNs and secure Web communications.

The usage of UDP port 500 for isakmp is discussed in RFC2408 (Maughan).

4.5. Attack Mechanism

The isakmp protocol uses UDP port 500 to exchange information to en able secure keys to be established which will themselves be used to secure some other communication path (possibly a TCP socket conveying HTTP data).

Unsolicited activity using UDP port 500 *would be expected* in circumstances where secure facilities were o ffered by the destination system (e.g. VPNs). A system *not* offering any such services *would not expect* any such unsolicited activity, therefore this activity is considered to be reconnaissance activity.

It is assumed that if UDP port 500 responded and wa s used to implement isakmp then the source system would follow up with a probe or attack on the secure service accessed via the key negotiated with isakmp. The UDP port 500 probe failed and therefore no subsequent activity was attempted.

4.6. Correlation

In archives.neophasis.com/archives/incidents/2000 -12/0110.html Greg Woods describes probes on UDP port 500, his consensus is that it appears to be a reconnaissance probe to se e if the probed site is providing VPN facilities.

In archives.neophasis.com/archives/incidents/2000 -12/0114.html TJ Jablonowski states that this occurrence could be the result of a Windows 2000 system (perhaps unknowingly to it's user) attempting to establish a secure connection with an Internet site. However, this is not believed to be the case here because the full firewall logs were checked for other instance of the 3 source IP addresses encountered above and no such occurrences were found.

In www.incidents.org/archives/intrusions/msg01092.html on 18 July 2001 Vicki Irwin refers to t he second reference above and observes that recent Code Red instances have been seen to precede their TCP port 80 activity by probing UDP port 500 a few seconds before. It is unlikely that this instance is a variation of Code Red as it was detected back i n December 2000.

Various Penetration Test software tools (e.g. PGP's CyberCop Scanner) test to see if target systems are vulnerable to attack on UDP port 500. Such penetration tests can cause IDSs and firewalls to report potential intrusions.

4.7. Evidence of active targeting

There is no evidence of active targeting.

4.8. Severity

The severity formula explained in Detect 1 above.

Criticality: 5, the firewall *appears* to be the target (due to NAT).

Lethality: 2, this appears to be a reconnaissance attempt.

System countermeasures: 4, the Operating System should reply with 'port Unreachable' as the system does not listen on UDP port 500.

Network countermeasures: 5, the traffic was blocked by the firewall.

(5 + 2) - (4 + 5) = -2

4.9. Defensive Recommendations

None. The 'attack' was silently blocked by the firewall.

4.10. Multiple choice test question

If packets are observed going to port 500 on a system what additional data is particularly useful in diagnosing the cause of the 'attack'?

a. full fidelity data from the firewall showing port 80 packets a short time after the port 500 traffic.
b. the data content of the packets.
c. all data to or from the system in question.
d. logs from all firewalls on local networks.

Answer: a

5. Detect 5

Raw data taken from syslog log files ge nerated by FireWall, only those lines relevant to the detect are shown (the large quantity of lines shown is required to perform TCP retransmission analysis).

```
Apr 16 05:54:47 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:54:48 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:54:50 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:54:52 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:54:58 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:54:59 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:55:08 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:55:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:55:32 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:55:33 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:56:20 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:56:21 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 05:57:56 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17123] ->[62.188.143.198/113] l=0 f=0x2.
```

```
Apr 16 05:57:57 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/17124] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:12:37 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/22945] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:07 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23157] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:22 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/22945] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:28 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23157] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:29 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:32 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:33 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:34 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:38 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:40 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:50 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:52 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:13:52 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23157] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:07 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:10 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/22945] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:10 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:14 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:17 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:17 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:29 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:41 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23157] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:14:52 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:15:02 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:15:04 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:15:40 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:15:46 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/22945] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:00 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:03 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:05 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:06 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:11 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:14 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:16 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
```

```
Apr 16 06:16:16 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23157] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:17 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:21 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:22 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:23 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:29 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:31 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:35 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:37 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:38 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23158] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:40 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23159] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:45 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:47 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:49 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:16:58 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:17:13 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:17:16 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/23868] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:17:34 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:17:35 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:17:46 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:18:01 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:19:09 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24616] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:19:11 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24617] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:19:22 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24620] ->[62.188.143.198/113] l=0 f=0x2.
Apr 16 06:19:37 <firewall> FILTER: Remote access filter blocks: TCP PPP
[216.127.64.117/24641] ->[62.188.143.198/113] l=0 f=0x2.
Apr 30 01:30:31 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3809] ->[62.188.135.50/113] l=0 f=0x2.
Apr 30 01:30:34 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3809] ->[62.188.135.50/113] l=0 f=0x2.
Apr 30 01:30:40 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3809] ->[62.188.135.50/113] l=0 f=0x2.
Apr 30 01:30:56 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3812] ->[62.188.135.50/113] l=0 f=0x2.
Apr 30 01:30:59 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3812] ->[62.188.135.50/113] l=0 f=0x2.
Apr 30 01:31:05 <firewall> FILTER: Remote access filter blocks: TCP PPP
[195.82.124.160/3812] ->[62.188.135.50/113] l=0 f=0x2.
```

The above raw data was then translated by a custom written shell script on the UNIX system where the syslog logs are logged and passed into a spreadsheet to format the data to produce the following (some lines are out of sequence so they can be grouped by

source port number, the large quantity of lines shown is required to perform TCP retransmission analysis):

```
Date     Time     Pro Src_IP         SrcP  Dst_IP         DstP Len Flags_etc Secs
16/04/01 05:54:47 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
16/04/01 05:54:50 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:00:03
16/04/01 05 :54:58 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:00:08
16/04/01 05:55:08 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:00:10
16/04/01 05:55:32 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 05: 56:20 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 05:57:56 TCP 216.127.64.117 17123 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 05:54:48 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
16/04/01 05 :54:52 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:00:04
16/04/01 05:54:59 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:00:07
16/04/01 05:55:09 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:00:10
16/04/01 05: 55:33 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 05:56:21 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 05:57:57 TCP 216.127.64.117 17124 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:12:37 TCP 216.127.64.117 22945 62.188.143.198  113   0 ......S.
16/04/01 06:13:22 TCP 216.127.64.117 22945 62.188.143.198  113   0 ......S.
0:00:45
16/04/01 06:14:10 TCP 216.127.64.117 22945 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06: 15:46 TCP 216.127.64.117 22945 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:13:07 TCP 216.127.64.117 23157 62.188.143.198  113   0 ......S.
16/04/01 06:13:28 TCP 216.127.64.117 23157 62.188.143.198  113   0 ......S.
0:00:21
16/04/01 06 :13:52 TCP 216.127.64.117 23157 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 06:14:41 TCP 216.127.64.117 23157 62.188.143.198  113   0 ......S.
0:00:49
16/04/01 06:16:16 TCP 216.127.64.117 23157 62.188.143.198  113   0 ......S.
0:01:35

16/04/01 06:13:29 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
16/04/01 06:13:33 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:00:04
16/04/01 06:13:38 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:00:05
16/04/01 06 :13:50 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:00:12
16/04/01 06:14:14 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 06:15:02 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06: 16:38 TCP 216.127.64.117 23158 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:13:32 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
16/04/01 06:13:34 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:00:02
```

```
16/04/01 06 :13:40 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:00:06
16/04/01 06:13:52 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:00:12
16/04/01 06:14:17 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:00:25
16/04/01 06: 15:04 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:00:47
16/04/01 06:16:40 TCP 216.127.64.117 23159 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:14:07 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
16/04/01 06 :14:10 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:00:03
16/04/01 06:14:17 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:00:07
16/04/01 06:14:29 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:00:12
16/04/01 06: 14:52 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:00:23
16/04/01 06:15:40 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06:17:16 TCP 216.127.64.117 23868 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:16:00 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
16/04/01 06:16:05 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:00:05
16/04/01 06:16:11 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:00:06
16/04/01 06: 16:21 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:00:10
16/04/01 06:16:45 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 06:17:34 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:00:49
16/04/01 06:1 9:09 TCP 216.127.64.117 24616 62.188.143.198  113   0 ......S.
0:01:35

16/04/01 06:16:03 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
16/04/01 06:16:06 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:00:03
16/04/01 06: 16:14 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:00:08
16/04/01 06:16:23 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:00:09
16/04/01 06:16:47 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 06:1 7:35 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06:19:11 TCP 216.127.64.117 24617 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:16:16 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
16/04/01 06: 16:17 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:00:01
16/04/01 06:16:22 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:00:05
16/04/01 06:16:35 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:00:13
16/04/01 06:1 6:58 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:00:23
16/04/01 06:17:46 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06:19:22 TCP 216.127.64.117 24620 62.188.143.198  113   0 ......S.
0:01:36

16/04/01 06:16:29 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
```

```
16/04/01 06:16:31 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:00:02
16/04/01 06:16:37 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:00:06
16/04/01 06:1 6:49 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:00:12
16/04/01 06:17:13 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:00:24
16/04/01 06:18:01 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:00:48
16/04/01 06:19 :37 TCP 216.127.64.117 24641 62.188.143.198  113   0 ......S.
0:01:36

30/04/01 01:30:31 TCP 195.82.124.160  3809 62.188.135.50   113   0 ......S.
30/04/01 01:30:34 TCP 195.82.124.160  3809 62.188.135.50   113   0 ......S.
0:00:03
30/04/01 01:3 0:40 TCP 195.82.124.160  3809 62.188.135.50   113   0 ......S.
0:00:06

30/04/01 01:30:56 TCP 195.82.124.160  3812 62.188.135.50   113   0 ......S.
30/04/01 01:30:59 TCP 195.82.124.160  3812 62.188.135.50   113   0 ......S.
0:00:03
30/04/01 01: 31:05 TCP 195.82.124.160  3812 62.188.135.50   113   0 ......S.
0:00:06
```

5.1. Source of trace

My (small business) network.

5.2. Detect was generated by

Firewall software causing syslog entries to be generated in response to violation of firewall access filters. See Detects above for details.

It should be noted that the dial-on-demand firewall was online for several minutes before the times when these packets were observed, it is *not* likely that any of this data was a remnant of previous use of the dynamically assigned IP address.

5.2.1. Firewall syslog format

As for the Detects above.

5.2.2. Filtered and processed data format

As for the Detects above.

5.3. Probability that source address was spoofed

Low.

These TCP packets are SYN packets attempting to establish a TCP connection. A response SYN/ACK would be routed to the source address. There would be little point spoofing the source address.

5.4. Description of the attack

This intercept appears to be an attempt at a DoS against SuSE Linux systems.

The 'ident' service identifies details (the system dependant 'user identifier') of the user of an established TCP connection. A default configuration of the

ident service in SuSE Linux systems allows a remote attacker to conduct a DoS attack.

For further details refer to **CVE-1999-0746** at cve.mitre.org and www.networkice.com/advice/Exploits/Ports/113/default.htm .

Further details of the ident protocol can be found in RFC 1413 (St, Johns).

BugtraqID: 587 also refers to this vulnerability.

5.5. Attack Mechanism

BugtraqID: 587 provides details of this vulnerability stating the DoS works by exhausting available memory on the target system by causing th e system to start multiple ident processes to respond to the large number of connections requests sent by the attacker.

The first group of intrusions came from 216.127.64.117 (DNS name not found) and we observe TCP retransmissions (the source port remains the same for small groups in detected packets and the time between those packets seems to show exponential backoff). The backoff timings could be used to detect the origin of the IP stack in use in the source's operating system, these seem to consistently double from 3 seconds until 96 seconds after which the originating IP software gives up attempting to make the connection.

The first observed occurrence came from source port 17123 and the next from port 17124, use of sequential ports in this way is oft en caused by program code intentionally looping to repeat an operation. Later we see ports 13257, 23158 and 23159 which again indicates automation rather than separate manually initiated activity.

Even later activity (source ports 24616, 24617, 24620 and 24641) indicate that other activity was occurring on the source system between those ports being opened for use. The activity could be similar SYN attempts to other destination addresses or some other activity, perhaps local, on the source system.

The two SYN attempts observed on 30 Apr 2001 from source IP address 195.82.124.160 (DNS name not found) show a different TCP retransmission characteristic, it is probable that the source operating system is different from that used above.

The above fine detail gives us intelligence which enables us to guess that the first system was possibly carrying out simultaneous attacks against multiple destination addresses of which we were just one.

5.6. Correlation

This vulnerability is referred to as 'SuSE identd Denial of Service Attack' and was attributed to Hendrik Scholz <hendrik@SCHOLZ.NET> on 14 Aug 1999.

5.7. Evidence of active targeting

This is *probably not* active targeting against a specific target system, it is more likely to be evidence of a scan over several destinat ion IP addresses. The use of three widely different source IP addresses over several months indicates reconnaissance scans being performed.

5.8. Severity

The severity formula explained in Detect 1 above.

Criticality: 5, the firewall *appears* to be the target (due to NAT).

Lethality: 4, it is a DoS.

System countermeasures: 5, the Operating System will send a TCP Reset because we are not running the ident service.

Network countermeasures: 5, the traffic was blocked by the firewall.

(5 + 4) - (5 + 5) = -1

5.9. Defensive Recommendations

None. The 'attack' was silently blocked by the firewall.

5.10. Multiple choice test question

How might causing a target system to run a large number of instances of a service program (e.g. identd, telnetd) cause a DoS?

a. By causing the t arget system to run out of disk space.
b. Because the target system will change its IP address.
c. Because only three instances of any service are allowed.
d. By exhausting the available memory on the target system.

Answer: d

Assignment 2 - Describe the State of Intrusion Detection

Simplified Analysis Tools

1. Introduction

This assignment was undertaken around the time when SANS NewsBites Vol. 3 Num. 37 were published. A tutorial was included from Bill Murray and other NewsBites editors titled "Protection of Home/SOHO Systems with Persistent Connections and IP Addresses". The tutorial notes that "Once penetrated, they [Home/SOHO systems] become a hazard to their neighbors. As their numbers increase, they become a threat to the health of the net." (Murray, p. 1).

The author of this paper operates a SOHO environment and has developed various analysis tools that can be used to assist protecting Home/SOHO systems. These tools are presented here and have been made available for others to use and develop f urther.

Some major manufacturers are spending considerable effort developing 'correlation engines' that will enable their customers to collate and process data from many sources. Small and medium sized enterprises (SMEs) often cannot afford these product s and therefore there is a requirement for an equivalent product for SMEs. This paper provides some tools that can be used by those operating in the SME range of businesses.

2. Objective

To produce analysis tools which will run on less expensive UNIX based c omputer systems. Some UNIX based systems have neither language compilers nor very fast processors. The objectives in writing these tools were:

1. To enable the tools to run on systems with limited run -time commands;
2. To enable the tools to run as quickly as possible.
3. To develop tools which will generate data to 'link graphs'.

A limited set of tools were used … those that come with a standard UNIX system, notably:

Name	Derivation of name	Purpose
ksh	Named after David Korn, one of its authors	A command inter preter.
awk	Named after Messrs. Aho, Weinberger and Kernighan	Match patterns of given data and possibly process them in various ways.
cat	Concatenate files	Join input files together to produce output.
grep	Globally search for regular expression and print it	Select patterns of given data, do not process them.
Sed	Stream Editor	Edit the input data to produce different output data, capable of working on *very* large volumes of data compared to some text editors.
Sort	The program sorts data!	To sort given input data into specified order(s) and produce sorted output.

3. Approach

Note: Data extracted from the analyst's UNIX system is shown here in Courier font, coloured blue for easier reading.

Snort produces data in different formats depending on the type of data. Snort alert files are in the following format:

```
<header lines>
MM/DD-hh:mm:ss.<fract>  [**] <text> [**] <src_ip>:<src port>   -> <dst ip>:<dst port>
```

For example:

```
********************************************************
                Snort Alert Report at Mon Jul  2 00:05:14 2001
********************************************************
07/01-00:00:30.477266  [**] UDP SRC and DST outside network [**] 169.254.161.0:1
37 -> 130.132.143.43:137
```

Snort scan files are in the following format:

```
<header lines>
MMM DD hh:mm:ss <src_ip>:<src port>   -> <dst ip>:<dst port>
```

For example:

```
********************************************************
                Snort Scan Report at Mon Jul  2 00:10:43 2001
********************************************************
Jul  1 00:12:47 MY.NET.6.45:7000   -> 129.240.86.35:7001 UDP
```

Snort OOS (Out Of Specification) files are in the following format:

```
<header lines>
MM/DD-hh:mm:ss.<fract> <src_ip>:<src port>   -> <dst ip>:<dst port>
<additional data lines ...>
<separator lines>
```

For example:

```
Initializing Network Interface ep0
snaplen = 68
Entering readback mode....
07/01-00:36:54.628555 63.254.9.59:32899   -> MY.NET.70.97:11055
TCP TTL:120 TOS:0x0 ID:60416  DF
21**R*AU Seq: 0x339E9EF3   Ack: 0xB3E894EE   Win: 0x623
F3 1A 1F 50 CD  52 70 3A 5A E3                    ...P.Rp:Z.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Development of tools to process these differing formats of data would be simpler if the formats were the same. Can these formats be converted into a single format? Some formats have certain data that other formats lack e.g. the fraction of a second field that is present in alert and oos files but lacking in scans files. The approach was taken that data pertaining on event be presented on one line and that any field not having a value in a particular format would be left blank.

The scripts used here process the raw Snort data into a consistent comma separated format:

```
Date,Time,Fraction_of_Second,Src_IP,Src_Port,Dst_IP,Dst_Port,Other
```

where Other varies according to the type of data being processed.

Where a data field is not present a null field is generated. Beware that there may be commas in the *final* field, typically where the data is the ASCII representation of the data (payload) part of a packet in the OOS data.

The scripts used for this analysis are available from the author's web site in zipped tar format or tar format for those who may wish to use them for their own purposes.

3.1. Phase 1 - Translating the data formats

The Snort alert files are processed with a Korn shell script named 'alert.ksh':

```
#!/bin/ksh
# Process Snort alert files to produce lines of t  he form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Note: 'portscan status' lines are removed as their data is
#       contained within a following 'end of portscan' line.
#
# Usage cat alert_file[s] | alert.ksh >outputfile
#
# e.g. cat aler t.0107*.txt | alert.ksh >alert.txt

grep '^[01][0 -9]/[0-9][0-9]-' | grep  -v ' portscan status from ' |   \
    sed  -f alert.sed
exit 0
```

The script uses grep to find the lines containing 'MM/DD -' (i.e. *not* header lines), passes those lines to another instance of grep (with the '-v' flag) which searches for lines *not* including the pattern ' portscan status from ' and then passes the resulting data to sed which processes it according to the sed pattern file named 'alert.sed':

```
s/,//g
s/-/,/
s/\./,/
s/\([^ ]*\) \(.....*\*\]\) \(.*\):\(.*\) -> \(.*\):\(.*\)/\1,\3,\4,\5,\6,\2/
s/\([^ ]*\) \(.*\) from \([^ ]*\) \((.*\)/\1,\3,,,,\2 \4/
s/\([^ ]*\) \(.*\) from \([^ ]*\)\(: .*\)/\1,\3,,,,\2 \4/
s/ \[\*\*\]//g
s/, /,/
```

The sed script works as follows: line 1 removes any commas from the input data; line 2 converts *the first* hyphen into a comma; line 3 converts *the first* fullstop (decimal points) into a comma; line 4 searches for the text between the [**] markers, the IP addresses and ports and reorders them into the requir ed format with commas between them; line 5 searches for and processes some of the portscan lines; line 6 searches for and processes other portscan lines; line 7 removes the '[**]' strings; and line 8 removes spaces after commas.

The Snort scans files ar e processed with a Korn shell script named 'scans.ksh':

```
#!/bin/ksh
# Process Snort scans files to produce lines of the form ...
```

```
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Usage cat scans_file[s] | scans.ksh >outputfile
#
# e.g. cat scans.0107* .txt | scans.ksh >scans.txt

grep '^[A-Z][a-z][a-z] ' | sed  -f scans.sed
exit 0
```

The script uses grep to find the lines containing a capital letter followed by two lower case letters at the beginning of a line (i.e. *not* header lines) and passes those lines to sed which processes it according to the sed pattern file named 'scans.sed':

```
s/^\(...\)  /\1 0/
s/^Jan /01\//
s/^Feb /02\//
s/^Mar /03\//
s/^Apr /04\//
s/^May /05\//
s/^Jun /06\//
s/^Jul /07\//
s/^Aug /08\//
s/^Sep /09\//
s/^Oct /10\//
s/^Nov /11\//
s/^Dec /12\//
s/\([^ ]*\) \([^ ]*\) \([^ ]*\):\([^ ]*\) -> \([^ ]*\):\([^ ]*\)
/\1,\2,,\3,\4,\5,\6,/
```

The sed script works as follows: line 1 searches for lines where the day of the month is less than ten and inserts a leading '0'; lines 2 -13 convert three letter month names into 2 digit month numbers; and line 14 formats the remaining data with commas between the fields.

The Snort OOS files are processed with a Korn shell script named 'oos.ksh':

```
#!/bin/ksh
# Process Snort oos files to produce lines of the   form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Usage cat oos_file[s] | oos.ksh >outputfile
#
# e.g. cat oos.0107*.txt | oos.ksh >oos.txt

ALL=
while read LINE
do
    if test ! "$LINE"
    then
        test "$ALL" && echo "$ALL"
        ALL=
    else
        case "$LINE" in
        [01][0-9]/* ) ALL="$ALL $LINE" ;;
        * ) test "$ALL" && ALL="$ALL $LINE" ;;
        esac
    fi
done | sed  -f oos.sed
exit 0
```

The script joins lines to form a single line for each event (ignoring blank lines and separator lines). The resultant data is passed to sed which processes it according to the sed pattern file named 'oos.sed':

```
s/^ //
s/-/,/
s/\./,/
s/\([^ ]*\) \([^ ]*\):\([^ ]*\) -> \([^ ]*\):\([^ ]*\) /\1,\2,\3,\4,\5,/
s/ [ ]*/ /g
```

The sed script works as follows: line 1 removes a space at the start of a line; line 2 changes *the first* hyphen into a comma; line 3 converts *the first* fullstop (decimal points) into a comma; line 4 formats the remaining data with commas between the fields; and line 5 reduces multiple spaces to just one space.

The scans, alerts and oos data having been formatted into a consistent format can now be processed to produce useful data and statistics for further analysis.

3.2. Phase 2 - Generating 'top_talkers' data

One of the useful statistics required when analysing IDS data is detecting which IP addresses are being reported most by the IDS. The IP address causing the most events to be produced by an IDS is called the 'top talker'. The 'top talkers' can be ranked into a list. A similar concept is the 'top destination' where the most frequently targeted IP address is detected. Scripts were developed to generate 'top talker' and 'top destinations' data.

The output from the formatting scripts is processed to generate 'top talkers' list with a Korn shell script named 'top_talk.ksh':

```
#!/bin/ksh
# Generate list of top talkers from lines of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# IP COUNT
#
# Usage top_talk.ksh <inputdata  >outputfile
#
# e.g. top_talk.ksh <alert.txt >top_talk.txt
# e.g. cat alert.txt oos.txt scans.txt | top_talk.ksh >top_talk.txt

awk -F, '{print $4}' | awk -f count.awk | sort -nr -k 2
exit 0
```

The script joins uses two instances of the 'awk' program. The f irst awk program outputs just field 4 (source IP address) from each input line (the ' -F,' flag causes awk to use ',' as a field seperator) and passes these (single field) lines to the second instance of awk which processes it according to the awk program i n the file named 'count.awk':

```
{
    for (i=1; i<=tot; i++) {
        if (value[i] == $0) {
            count[i]++
            break
        }
    }
    if (i > tot) {
        value[i] = $0
        count[i] = 1
        tot++
    }
}
END {
    for (i=1; i<= tot; i++) {
        print value[i], count[i]
```

```
    }
}
```

The awk program works as follows: the for loop searches for the current line of text in an awk data array (which is empty at the start of the program); if the for loop finds the text in the array then it increments a counter of how many times the text occurs in the data. If the text is *not* found in the array then a new line is added to the array with the text in question and an occurrence count of 1. The lines beginning at the ' END' statement tell the awk program what to do when all data has been processed; the program loops through all the entries in the array and prints out the text and the occurrences count.

A similar script is used to generate a 'top destination' list with a Korn shell script name d 'top_dest.ksh':

```
#!/bin/ksh
# Generate list of top destinations from lines of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# IP COUNT
#
# Usage top_dest.ksh <inputdata >outputfile
#
# e.g. top_dest. ksh <alert.txt >top_dest.txt
# e.g. cat alert.txt oos.txt scans.txt | top_dest.ksh >top_dest.txt

awk -F, '{print $6}' | awk  -f count.awk | sort  -nr -k 2
exit 0
```

The script works in the same way as the 'top_talk' script except that it outputs just field 6 (destination IP address) from each input line for passing to the count.awk program.

3.3. Phase 3 - Generating IP address specific data

Another useful toolset is one to extract data relating to specified IP addresses. The 'ip.ksh' script selects events concerning an IP address (the address can be either the source or destination):

```
#!/bin/ksh
# Select lines with destination IP address, lines are of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# DATE,TIME,FR ACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Usage ip.ksh address <inputfile >outputfile
#
# e.g. ip.ksh 1.2.3.4 <alert.txt >ip.txt
# e.g. cat alert.txt oos.txt scans.txt | ip.ksh 1.2.3.4 >ip.txt

USAGE='Usage: ip.ksh address <inputfile >outputfile'
test $# != 1 && echo $USAGE >&2 && exit 1

grep -e "^[^,]*,[^,]*,[^,]*,$1,"  -e "^[^,]*,[^,]*,[^,]*,[^,]*,[^,]*,$1,"
exit 0
```

The 'ip_src.ksh' script selects events with a specified source IP address:

```
#!/bin/ksh
# Select lines with source IP address, lines are of th  e form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
```

```
# output lines are of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Usage ip_src.ksh address <inputfile >outputfile
#
# e.g. ip_src.ksh 1.2.3.4 <alert.txt >ip_src.txt
# e.g. cat alert.txt oos.txt scans.txt | ip_src.ksh 1.2.3.4 >ip_src.txt

USAGE='Usage: ip_src.ksh address file[s] >outputfile'
test $# != 1 && echo $USAGE >&2 && exit 1

grep "^[^,]*,[^,]*,[^,]*,$1,"
exit 0
```

The 'ip_dest.ksh' script selects events with a spec ified destination IP address:

```
#!/bin/ksh
# Select lines with destination IP address, lines are of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# Usage ip_dest.ksh address <inputfile >outputfile
#
# e.g. ip_dest.ksh 1.2.3.4 <alert.txt >ip_dest.txt
# e.g. cat alert.txt oos.txt scans.txt | ip_dest.ksh 1.2.3.4 >ip_dest.txt

USAGE='Usage: ip_dest.ksh address <inputfile >outputfile'
test $# != 1 && echo $ USAGE >&2 && exit 1

grep "^[^,]*,[^,]*,[^,]*,[^,]*,[^,]*,$1,"
exit 0
```

These tools can be combined with those that follow to select comprehensive statistics sets.

3.4. Phase 3 - Generating top port usage data

Another useful statistic might be that of the most prolific port(s) in use. A script called 'top_port.ksh' was developed to process the event data to generate top (source or destination) port usage:

```
#!/bin/ksh
# Generate list of top port usage for a given IP address, input in the form ...
# DATE,TIME,FRA CT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are in the form ...
# IP PORT
#
# Usage top_port.ksh address <inputfile >outputfile
#
# e.g. top_port.ksh 1.2.3.4 <alert.txt >top_port.txt
# e.g. cat alert.txt oos.txt scans.txt | top_port.ksh 1.2.3.4 >t   op_port.txt

ADDR=$1

awk -F, "{
    if (\$4 == \"$ADDR\") { printf \"%s,%d\\n\", \"$ADDR\", \$5 }
    if (\$6 == \"$ADDR\") { printf \"%s,%d\\n\", \"$ADDR\", \$7 }
}" | awk -f count.awk | sort  -nr -k 2
exit 0
```

The awk program in the above script looks fo r the selected IP address in either the source or destination address field and outputs the *corresponding* port number.

The 'top_portd.ksh' script processes the event data to generate top destination port usage:

```
#!/bin/ksh
# Generate list of top dest por  t usage
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# PORT COUNT
#
# Usage top_portd.ksh <inputfile >outputfile
#
# e.g. top_portd.ksh <alert.txt >top_portd.txt
# e.g. cat alert.txt oos.txt scans.txt | top_portd.   ksh >top_portd.txt

awk -F, '{print $7}' | awk  -f count.awk | sort  -nr -k 2
exit 0
```

The 'top_portd.ksh' script processes the event data to generate top destination port usage:

```
#!/bin/ksh
# Generate list of top dest port usage
# DATE,TIME,FRACT,SRCIP,SRCP  ORT,DSTIP,DSTPORT,TEXT
#
# output lines are of the form ...
# PORT COUNT
#
# Usage top_portd.ksh <inputfile >outputfile
#
# e.g. top_portd.ksh <alert.txt >top_portd.txt
# e.g. cat alert.txt oos.txt scans.txt | top_portd.ksh >top_portd.txt

awk -F, '{print  $7}' | awk  -f count.awk | sort  -nr -k 2

exit 0
```

An example of use of this script is to find the destination ports which are most targeted in packets containing OOS data:

```
$ top_portd.ksh <oos.txt | head  -5
111 557
80 470
25 182
6346 52
1214 41
```

This clearly shows that the selected OOS data (in the 'oos.txt' file) contains 557 events where the target port was 111 (portmapper). The second most prevalent targeted port was 80 (http).

3.5. Phase 4 - Generating 'link graph' data

The final set of tools that have bee n developed to date are used to assist generation of 'link graphs'. Link graphs are designed to show, in a graphical form, the data flow of selected analysed data.

The first link script 'links.ksh' produces a list of links to or from a named IP address:

```
#!/bin/ksh
# Generate data for generating a link graph, input data is of the form ...
# DATE,TIME,FRACT,SRCIP,SRCPORT,DSTIP,DSTPORT,TEXT
#
# output data is of the form ...
# IP1 PORT1 DIR IP2 PORT2 COUNT
#
# Usage links.ksh ADDR <inputfile >outputfile
```

```
#
# e.g. links.ksh 1.2.3.4 <oos.txt >links.txt
# e.g. cat scans.txt oos.txt | links.ksh 1.2.3.4 >links.txt

IP=$1

awk -F, "{
    if (\$4 == \"$IP\" && \$6 != \"\") { print \$4, \$5, \"to\", \$6, \$7 }
    if (\$6 == \"$IP\" && \$4 != \"\") { print \$6, \$7, \"fm\", \$4, \$5 }
}" | awk -f count.awk | sort  -nr -k 6
exit 0
```

An example of use of this script is shown below:

```
$ ./links.ksh 24.159.128.162 <alert.txt >temp1
$ cat temp1
24.159.128.162 3933 to MY.NET.112.141 27374 4
24.159.128.162 3897 to MY.NET.112.1  03 27374 4
...
24.159.128.162 3869 to MY.NET.112.74 27374 3
24.159.128.162 3861 to MY.NET.112.66 27374 3
24.159.128.162 3855 to MY.NET.112.60 27374 3
24.159.128.162 3850 to MY.NET.112.54 27374 3
24.159.128.162 3822 to MY.NET.112.25 27374 3
24.159.128.162 3 822 fm MY.NET.112.25 27374 3
24.159.128.162 3793 to MY.NET.111.251 27374 3
24.159.128.162 3737 to MY.NET.111.193 27374 3
24.159.128.162 3730 to MY.NET.111.186 27374 3
24.159.128.162 3676 to MY.NET.111.130 27374 3
24.159.128.162 3676 fm MY.NET.111.130 27374   3
...
```

The output is sorted in the script to show the highest port usage first. The lines shown in bold highlight two-way traffic, 3 packets were seen from the selected IP address **to** another address and 3 packets were seen to the selected IP address **from** another address.

The data produced by the previous script can be used as input to the next two scripts, the first one is named 'linkg.ksh', its purpose is to generate a link graph (albeit in text format!):

```
#!/bin/ksh
# Generate (NOT sortable) link graph  , input data is of the form ...
# IP1 PORT1 DIR IP2 PORT2 COUNT
#
# Usage linkg.ksh LOW HIGH <inputfile >outputfile
#
# e.g. linkg.ksh 2 10 <oos.txt >linkg.txt
# e.g. cat scans.txt oos.txt | linkg.ksh 2 10 >linkg.txt

# 1 .. LO are the values to be marked   as LEAST occurring (i.e. " --->")
# HI ..   are the values to be marked as MOST  occurring (i.e. ">>>>")
# (other values are marked as medium frequency occurring) (i.e. "   ->->")

LO=$1
HI=$2

sort -k 1,1 -k 4,4 -k 2,2 -k 5,5 -k 3,3 -k 6,6 | awk "{
    if (\$3 == \"to\") {
        if (\$6 <= $LO)            { ptr =  \"--->\" }
        else if ( \$6 >= $HI)    { ptr =  \">>>>\" }
        else                        { ptr =  \"->->\" }
    } else {
        if (\$6 <= $LO)            { ptr =  \"<---\" }
        else  if (\$6 >= $HI)    { ptr =  \"<<<<\" }
        else                        { ptr =  \"<-<-\" }
    }
```

```
    if (\$1 != last_ip1) { ip1 =  \$1 } else { ip1 =  \"\" }
    if (\$4 != last_ip2) { ip2 =  \$4 } else { ip2 =  \"\" }
    last_ip1= \$1
    last_ip2= \$4
    format= \"%-15s %5s %s %6d %s % -5s %-15s\n\"
    printf format, ip1,  \$2, ptr, \$6, ptr, \$5, ip2
}"
exit 0
```

An example of use of this script is shown below:

```
$ linkg.ksh 2 4 <temp1
24.159.128.162   3676 < -<-      3 <-<- 27374 MY.NET.111.130
                  3676 ->->      3 ->-> 27374
                 3730  ->->      3 ->-> 27374 MY.NET.111.186
                 3737  ->->      3 ->-> 27374 MY.NET.111.193
                 3793  ->->      3 ->-> 27374 MY.NET.111.251
                 3624 < -<-      3 <-<- 27374 MY.NET.111.75
                 3624  ->->      3 ->-> 27374
                 3630  ->->      3 ->-> 27374 MY.NET.111.82
                 3638 < -<-      3 <-<- 27374 MY.NET.111.90
                 3897 >>>>      4 >>>> 27374 MY.NET.112.103
                 393 3 >>>>      4 >>>> 27374 MY.NET.112.141
                 3822 < -<-      3 <-<- 27374 MY.NET.112.25
                 3822  ->->      3 ->-> 27374
...
```

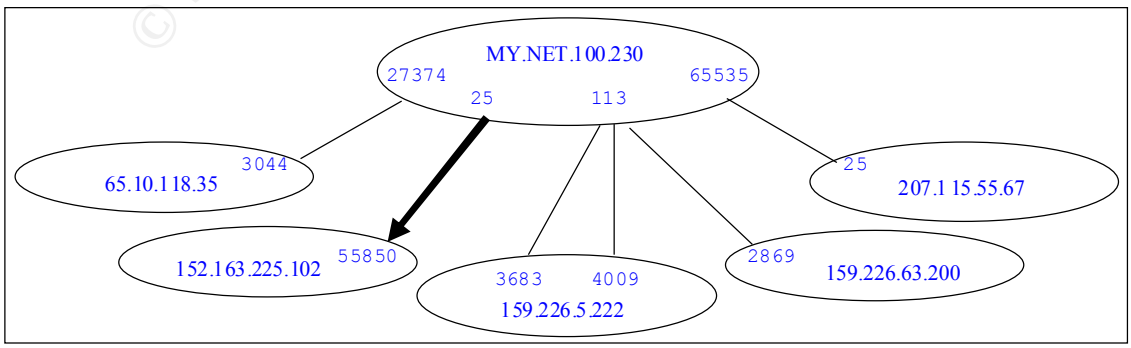
Note: The '--->' arrows are supposed to indicate *thin* arrows, the '->->' arrows are supposed to indicate *medium thickness* arrows and the '>>>>' arrows are supposed to indicate *thick* arrows. The thresholds of thin and thick are provided by arguments to the script.

Another example of use of these link data generation scripts is:

```
$ ./links.ksh MY.NET.100.230 <a  lert.txt
MY.NET.100.230 25 to 152.163.225.102 55850 15
MY.NET.100.230 65535 to 207.115.55.67 25 7
MY.NET.100.230 27374 to 65.10.118.35 3044 2
MY.NET.100.230 113 fm 159.226.63.200 2869 1
MY.NET.100.230 113 fm 159.226.5.222 4009 1
MY.NET.100.230 113 fm 159.2  26.5.222 3683 1

$ ./links.ksh MY.NET.100.230 <alert.txt | ./linkg.ksh 2 10
MY.NET.100.230     25 >>>>     15 >>>> 55850 152.163.225.102
                  113 < ---      1 <--- 3683  159.226.5.222
                  113 < ---      1 <--- 4009
                   113 <---      1 <--- 2869  159.226.63.200
                65535  ->->      7  ->-> 25    207.115.55.67
                27374  --->      2  ---> 3044  65.10.118.35
```

This data could be converted into a graphical link graph:



After the above script had been written it was realised that the output of the script is not as flexible as it could be because some lines do not contain all fields of data (e.g. they cannot be sorted). A second script named 'linkg2.ksh' was developed to produce flexib le output:

```
#!/bin/ksh
# Generate link graph (sortable output), input data is of the form ...
# IP1 PORT1 DIR IP2 PORT2 COUNT
#
# Usage linkg2.ksh LOW HIGH <inputfile >outputfile
#
# e.g. linkg2.ksh 2 10 <oos.txt >linkg2.txt
# e.g. cat scans.txt oos.txt |  linkg2.ksh 2 10 >linkg2.txt

# 1 .. LOW are the values to be marked as LEAST occurring (i.e. "  --->")
# HIGH ..  are the values to be marked as MOST  occurring (i.e. ">>>>")
# (other values are marked as medium frequency occurring) (i.e. "  ->->")

LO=$1
HI=$2

sort -k 1,1 -k 4,4 -k 2,2 -k 5,5 -k 3,3 -k 6,6 | awk "{
    if (\$3 == \"to\") {
        if (\$6 <= $LO)         { ptr =  \"--->\" }
        else if ( \$6 >= $HI)   { ptr =  \">>>>\" }
        else                       { ptr =  \"->->\" }
    } else {
        if (\$6 <= $LO)         { ptr =  \"<---\" }
        else if ( \$6 >= $HI)   { ptr =  \"<<<<\" }
        else                       { ptr =  \"<-<-\" }
    }
    format= \"%-15s %5s %s %6d %s % -5s %-15s\n\"
    printf format,  \$1, \$2, ptr, \$6, ptr, \$5, \$4
}"
exit 0
```

An example of use of this script is shown below:

```
$ ./linkg2.ksh 2 4 <temp1 | sort  -k 2 -k 3
24.159.128.162   3624  ->->      3 ->-> 27374 MY.NET.111.75
24.159.128.162   3624 < -<-      3 <-<- 27374 MY.NET.111.75
24.159.128.162   3630  ->->      3 ->-> 27374 MY.NET.111.82
24.159.128.162   3638 < -<-      3 <-<- 27374 MY.NET.111.90
24.159.128.162   3676  ->->      3 ->-> 27374 MY.NET.111.130
24.159.128.162   3676 < -<-      3 <-<- 27374 MY.NET.111.130
24.159.128.162   3730  ->->      3 ->-> 27374 MY.NET.111.186
24.159.128.162   3737  ->->      3 ->-> 27374 MY.NET.111.193
24.159.128.162   3793  ->->      3 ->-> 27374 MY.NET.111.251
24.159.128.162   3822  ->->      3 ->-> 27374 MY.NET.112.25
24.159.128.162   3822 < -<-      3 <-<- 27374 MY.NET.112.25
24.159.128.162   3850  ->->      3 ->-> 27374 MY.NET.112.54
24.159.128.162   3855  ->->      3 ->-> 27374 MY.NET.112.60
24.159.128.162   3861  ->->      3 ->-> 27374 MY.NET.112.66
24.159.128.162   3869  ->->      3 ->-> 27374 MY.NET.112.74
24.159.128.162   3897  >>>>      4 >>>> 27374 MY.NET.112.103
24.159.128.162   3933 >>>>      4 >>>> 27374 MY.NET.112.141
```

3.6. Phase 5 - Generating ad-hoc data

The benefit of constructed so many seemingly trivial scripts is that they may be used in conjunction with each other. Many combinations of the scripts were used in analysing the data in part 3 of this practical.

4. Conclusions

With the increase in Home/SOHO systems in use today the Internet community needs greater protection for such devices to avoid their being used as third pa rty agents by those with malicious intent. The owners of these systems may be law abiding and not intending to cause problems but with their systems connected to 'always on' communications technology they provide immense processing power which **will** be harnessed by others if the owners do not protect them.

Various vendors sell small firewall devices that are able to be configured with access/deny rules and produce log information. Various IDSs are available free of charge (e.g. Snort) which can be cheaply installed and configured. Both of these mechanisms can generate huge amounts of logging information, which is next to useless, if nobody examines it and actions their findings.

The aim of this paper has been to encourage thought about this problem and t o provide some simple tools that will help the analysis of log data.

Assignment 3 - "Analyse This"

The scenario for this report was that the author had been asked to provide a security audit for a University and had been provided with data from a Snort Intrusion Detection System using a fairly standard set of rules.

1. Executive Summary

Data for the period 1 -5 July 2001 (Sun-Thurs) was selected and analysed.

The logs showed that there was considerable anomalous activity within the network. This activity can cause problems in several ways: it may be malicious and illegal; it may also be slowing down genuine network activity.

There were 15188 occurrences of 'possible trojan server activity' which means that there is a considerable chance that rogue programs have been installed (intentionally or not) on systems within the University. These trojans may be using University systems to cause problems to other Internet users. The trojans may also be letting malicious personnel access to the University's sensitiv e data.

There were 2172 occurrences of the network being scanned. This activity is probably of no beneficial use. Data coming in to the University's networks from persistent scanning sites should be blocked.

Some systems have been seen to be running du bious software that may be leaking University data to outside systems.

It is recommended that the University develop a network architecture that allows the bulk of its systems to be behind firewalls that limit access from the Internet into the University.

It is suggested that a baseline of standards be devised and adopted for all operating systems in use. This baseline should be regularly updated to take advantage of various patches to overcome vulnerabilities and other operating systems shortfalls, as t hey become available.

To protect those systems which are used to explore out on the Internet (and those internally into which users install unknown software) it is suggested that regular checks are made to ensure that no rogue software is operating and th at the security baseline is intact. Tools such as CyberCop (Commercial) and Nessus (freely available) can be used for this task.

2. Files chosen for analysis

The files chosen were those for 1 st to 5th July 2001. One reason for this choice was that 1 st July 2001 was a Sunday that may lead to differences in occurrences compared to the other days (weekdays).

Various GCIA practicals were downloaded from the SANS site www.sans.org/giactc/gcia.htm to be searched for correlations. Other correlations were found by using search engines on the Internet (e.g. www.yahoo.com).

3. Top Five Prioritised Detects (by number of occurrences)

The detected alerts are summarised as follows:

Occurrences	Description
15188	Possible trojan server activity
8015	UDP SRC and DST outside network
2172	spp_portscan: End of portscan (TOTAL HOSTS:1 TCP:0 UDP:10)
1426	External RPC call
1294	connect to 515 from outside
690	Watchlist 000220 IL -ISDNNET -990517
386	SMB Name Wildcard
275	SYN-FIN scan!
217	Queso fingerprint
173	WinGate 1080 Attempt
157	Port 55850 tcp - Possible myserver activity - ref. 010313-1
108	SUNRPC highport access!
86	Watchlist 000222 NET -NCFC
71	NMAP TCP ping!
54	TCP SRC and D ST outside network
53	High port 65535 tcp - possible Red Worm - traffic
47	Null scan!
8	Attempted Sun RPC high port access
3	Back Orifice
2	Russia Dynamo - SANS Flash 28-jul-00
1	connect to 515 from inside
1	STATDX UDP attack
1	TCP SMTP Source Port traffic

3.1. Priority 1 Detect (Possible trojan server activity)

Of the 15188 alerts to possible trojan server activity, 12889 (85%) were *to* destination TCP port 27374 and the other 2299 (15%) were *from* source TCP port 27374. TCP port 27374 is commonly used by the Ramen worm and SubSeven trojan exploit.

The SubSeven trojan can be used by attackers to make use of the infected system for their own ends. Infected systems are also used to scan other systems to which they have access with the intent of planting the SubSeven virus on those systems, and so it spreads…

Without full fidelity logs of the packets that caused the alerts it is not possible to give an assured judgement on some of the alerts seen. An example of this is given in the following table where the data seen may be a TCP SYN from the 'attacker' (65.8.220.176) and a responding TCP RST from the system in the network being monitored (MY.NET.10.59). If this is the case then the MY.NET.10.59 system has *not* been compromised, it has *refused* the requested connection to TCP port 27374.

Date	Time	SRC IP	Src P	Dst IP	Dst P
02-Jul	19:50:29	65.8.220.176	4713	MY.NET.10.59	27374
02-Jul	19:50:29	MY.NET.10.59	27374	65.8.220.176	4713

Some of the alerts cause us to believe that logging may not have been comple te, packets may have been dropped. The example given here is of external system 65.8.220.176 appearing to scan internal addresses MY.NET.10.43 through MY.NET.10.57, although we do not see the

probe to MY.NET.10.54 we see the response from MY.NET.10.54 (no te the port number of 4708 on the external host, one more than the previously observed port used by the same host to probe MY.NET.10.53) which were hopefully TCP RSTs.

Date	Time	SRC IP	Src P	Dst IP	Dst P
02-Jul	19:50:29	65.8.220.176	4697	MY.NET.10.43	27374
02-Jul	19:50:29	65.8.220.176	4699	MY.NET.10.45	27374
02-Jul	19:50:29	65.8.220.176	4703	MY.NET.10.49	27374
02-Jul	19:50:38	65.8.220.176	4703	MY.NET.10.49	27374
02-Jul	19:50:29	65.8.220.176	4707	MY.NET.10.53	27374
02-Jul	19:50:38	65.8.220.176	4707	MY.NET.10.53	27374
02-Jul	19:50:29	MY.NET.10.54	27374	65.8.220.176	4708
02-Jul	19:50:30	MY.NET.10.54	27374	65.8.220.176	4708
02-Jul	19:50:30	MY.NET.10.54	27374	65.8.220.176	4708
02-Jul	19:50:29	65.8.220.176	4711	MY.NET.10.57	27374

In some cases repeate d alerts were raised between an external host and an internal host which gives greater cause for concern. However, examination of the Snort scan files gives greater insight into these scenarios. It appears that the external host repeatedly tried to conne ct with the internal host: repeated SYNs can be seen (again some packets seem to be missing):

Date	Time	SRC IP	Src P	Dst IP	Dst P
02-Jul	19:51:14	24.159.128.162	3822	MY.NET.112.25	27374
02-Jul	19:51:14	MY.NET.112.25	27374	24.159.128.162	3822
02-Jul	19:51:14	24.159.128.162	3822	MY.NET.112.25	27374
02-Jul	19:51:14	MY.NET.112.25	27374	24.159.128.162	3822
02-Jul	19:51:15	24.159.128.162	3822	MY.NET.112.25	27374
02-Jul	19:51:15	MY.NET.112.25	27374	24.159.128.162	3822

Date	Time	SRC IP	Src P	Dst IP	Dst P	Flags
02-Jul	19:51:14	24.159.128.162	3822	MY.NET.112.25	27374	SYN **S*****
02-Jul	19:51:15	24.159.128.162	3822	MY.NET.112.25	27374	SYN **S*****

There is a FAQ page at www.sans.org/ne wlook/resources/IDFAQ/subseven.htm that gives useful information about the ways to detect if your system has been infected by the SubSeven trojan.

Conclusion: The activity on TCP port 27374 was *probes* for install SubSeven trojan programs, none appear to have been found.

Recommendation: Block inbound TCP SYN traffic to port 27374 at the network's boundary firewall.

3.1.1. Correlations

CAN-1999-0660 and CAN-2000-0138 at www.mitre.org are both candidate vulnerabilities describing this type of activity.

PJ Goodwin makes various comments about TCP port 27374 scans in his GCIA paper at www.sans.org/y2k/practical/PJ_Goodwin_GCIA.doc , his report also refers to TCP port 27374 activity being primarily *probing* for installed SubSeven trojans, not necessarily finding and making use of them.

3.2. Priority 2 Detect (UDP SRC and DST outside network)

Packets should not be observed on a network if they are not either *from* an internal system or *to* an internal system. The only exception to this would be if the network in question was routing data between other networks (e.g. an ISP or a backbone Internet site). The site in question is a University and it is assumed that the Universi ty is *not intending* to route data for others.

The top five occurrences of this alert were as follows:

Src IP	Src P	Dst IP	Dst P	Dst Port	Occurrences
169.254.X.Y	137	A.B.C.D	137	NetBIOS Name	6038
169.254.161.0	137	130.132.143.43	137	NetBIOS Name	2927 *
169.254.161.0	137	130.132.143.42	137	NetBIOS Name	2873 *
63.250.213.26	1039	233.28.65.164	5779	IANA unassigned	1788
192.168.X.Y	137	63.240.138.21	137	NetBIOS Name	90

* These two figures are part of the 6038 in the first row.

The most prevalent trace i s of 6038 'NetBIOS Name service' (UDP port 137) packets from Class B network 169.254.X.Y to various destination addresses, including the two specific destination addresses shown in the table.

In his posting at http://www.shmoo.com/mail/fw1/mar01/msg01360.shtml , Ryan Vickmark states:

> In Windows 98 and Windows 2000 if a computer is set to use DHCP and is unable to locate a DHCP server it picks a random number from the 169.254.0.0/16 range.

A draft paper at http://www.ietf.org/proceedings/98dec/I -D/draft-ietf-dhc-ipv4-autoconfig-01.txt provides the clarification (Troll, p. 3):

> The address range to use MUST be "169.254/16", which is registered with the IANA as the LINKLOCAL net.

Conclusion: The above quote and the fact that the packets are 'NetBIOS name service' would seem to indicate the underlying problem.

Recommendation: Provide a default DHCP s erver to 'catch' Windows 98/2000 systems and give them a correct IP address for the network in question.

The fourth most prevalent trace is from UDP port 1039 at 63.250.213.26 to UDP port 5779 at 233.28.65.164. Addresses in the Class D range (224.0.0.0 to 239.255.255.255) are *multicast* addresses that are used wh en one source address communicates with *multiple* destinations with each packet. Addresses in the range 233.0.0.0 to 233.255.255.255 are assigned as the 'GLOP block' in RFC 3171 (Albanna, p. 2).

Packets addressed to multicast addresses are not explicitly destined for the internal network but when some process within the internal network 'joined' a multicast group the routers to, and source of, the multicast data were instructed to forward *copies* of the packets to the

internal network. They are therefore incorrectly classified as 'SRC and DST outside network'.

Conclusion: These 233.X.Y.Z addresses are *not* outside the SRC or DST range of addresses.

Recommendation: Add the address range 224.0.0.0 to 239.255.255.255 to the IDS's knowledge of internal addresses .

The fifth most prevalent trace is from UDP port 137 at various source addresses in the Class C range 192.168.X.Y. This address range is one of those referred to as a Private Address range detailed in RFC 1918 (Rekhter, p.4). Addresses in Private Address ranges should never be routed across the Internet, they should either be dropped at Internet routers or be subject to Network Address Translation by routers/firewalls.

One way to try to determine the source of these (non -multicast) packets would be to analyse a packet trace (e.g. from 'tcpdump') to find the MAC (hardware) address of the device originating these packets. If the MAC address is that of a router then packets will have to be traced on the 'other' side of the router until a source address that is not a router is detected. This device should then be examined for the source of the data.

Conclusion: Packets with either source or destination addresses in the ranges of Private IP Addresses defined in RFC 1918 (e.g. 192.168.X.Y) should not be forwarded by routers to the Internet, the packets should be restricted to their own private networks (e.g. a company or educational establishment). Any systems that need to be 'visible' to the In ternet from within such a private network should be subject the Network Address Translation so that the visible address in *not* within the Private Address range.

Recommendation: Filter Private Addresses form propagating through external routers/firewalls. Add the private address ranges to the IDS's knowledge of internal addresses.

3.2.1. Correlations

Russell Felton discusses similar 169.254.X.Y anomalies in his posting to http://www.theorygro up.com/Arhive/Argus/1999/msg00165.html .

Andrew Windsor makes various comments about 'UDP SRC and DST outside network' in his GCIA paper at http://www.sans.org/y2k/practical/Andrew_Wi ndsor_GCIA.doc , but in his analysis the DST address was always a Multicast address (224.x.y.z).

SANS handler Jeff Stsutzman report activity from 192.168.X.Y addresses to UDP port 90 & 138 in the SANS diary entry at www.sans.org/y2k/032900 -2030.htm.

3.3. Priority 3 Detect (Portscans)

The table of observed portscans is as follows (ranked by number of scans reported):

Src IP	Scans	Tot Hosts	Tot TCP	Tot UDP
199.183.24.194	90	90	90	
MY.NET.100.230	87	1302	113	1274
MY.NET.70.80	87	1295		1431

MY.NET.140.191	84	1451		1612
211.207.15.190	70	29885	30094	
24.66.152.186	58	58	59	
142.177.206.111	44	44		603
MY.NET.98.174	34	34	2247	
MY.NET.6.45	30	261		262
MY.NET.98.188	27	337	341	
213.118.56.46	25	6158	6266	
148.223.228.15	24	12837	13110	
61.222.34.170	24	10739	10994	
MY.NET.217.10	20	4671	1	5752

Although IP address 199.183.24.194 was reported most often by Snort for having performed scans, the figures for 211.207.15.190, 148.223.228.15 and 61.222.34.17 0 cause more concern because they scanned more hosts. The reason for this anomaly is that Snort has to decide when a scan has finished so that it can report the fact. Address 199.183.24.194 may have performed 90 scans but we can see from the number of ho sts scanned that each scan only scanned one host and only one [TCP] port on that host. Is that a scan? Is Snort reporting correctly? Looking at some of the lines in Snort's scan log files relating to this source address we see lines of the form:

```
07/01,00:56:44,,199.183.24.194,45255,MY.NET.253.43,25,SYN 21S***** RESERVEDBITS
07/01,10:15:51,,199.183.24.194,37893,MY.NET.253.41,25,SYN 21S***** RESERVEDBITS
```

which show that Snort was in fact seeing abnormal activity because the reserved bits in the TCP flags field (the higher order two bits) were set.

Both of the packets shown above were destined to port 25 (SMTP). The packets are crafted (generated by a probing program rather than a normal operating system's network stack) because the flags bits set ('21S') do not conform to TCP rules.

Site details:

nslookup: vger.kernel.org

whois: Server Name: NS.VGER.KERNEL.ORG
IP Address: 199.183.24.194
Registrar: NETWORK SOLUTIONS, INC.
Whois Server: whois.networksolutions.com

Conclusion: This address has been seen t o be carrying out malicious activity and is probably attempting to identify types of operating systems in use.

Recommendation: 1) That the address be blocked coming in to the network at the boundary firewall; and 2) that the administrators of the site be informed that their system has been observed scanning MY.NET.X.Y. If the address in question has been compromised by some third party then informing its owners will alert them to the fact and allow them to eradicate the problem.

Whilst only being reporte d by Snort for carrying out 70 scans, source address 211.207.15.190 scanned 29885 hosts! The table below shows some examples of the scans:

Date	Time	Src IP	Description
01-Jul	05:05:36	211.207.15.190	spp_portscan: PORTSCAN DETECTED (THRESHOLD 7 connectio ns in 2 seconds)
01-Jul	05:06:13	211.207.15.190	spp_portscan: End of portscan (TOTAL HOSTS:2509 TCP:2525 UDP:0)
01-Jul	05:06:29	211.207.15.190	spp_portscan: PORTSCAN DETECTED (THRESHOLD 7 connections in 2 seconds)
01-Jul	05:06:42	211.207.15.190	spp_portscan: End of portscan (TOTAL HOSTS:554 TCP:559 UDP:0)
01-Jul	05:06:44	211.207.15.190	spp_portscan: PORTSCAN DETECTED (THRESHOLD 7 connections in 2 seconds)
01-Jul	05:06:58	211.207.15.190	spp_portscan: End of portscan (TOTAL HOSTS:687 TCP:692 UDP:0)
01-Jul	05:07:00	211.207.15.190	spp_portscan: PORTSCAN DETECTED (THRESHOLD 7 connections in 2 seconds)
01-Jul	05:07:06	211.207.15.190	spp_portscan: End of portscan (TOTAL HOSTS:164 TCP:165 UDP:0)
01-Jul	05:07:09	211.207.15.190	spp_portscan: PORTSCAN D ETECTED (THRESHOLD 7 connections in 2 seconds)
01-Jul	05:07:39	211.207.15.190	spp_portscan: End of portscan (TOTAL HOSTS:2400 TCP:2414 UDP:0)

Examining Snort's scan log files for the period 01 -Jul 05:06:29 - 05:06:42 during which Snort reported that this source address had scanned 554 hosts and 559 TCP ports we see just 96 records, (each of which was on 01 -Jul, from the above address, to TCP port 21 (FTP Control), with just the SYN flags set). The log appears to be incomplete because we see increasing port numbers on the source system probing increasing IP addresses in the MY.NET range but there are gaps in each which match (i.e. if the source port jumps by 3 then the destination address also jumps by 3). The logged data shows that the scan covered addresses in the range MY.NET.145.2 to MY.NET.145.248 which, allowing for missed data, seems to be a complete scan of the MY.NET.145.1 to MY.NET.145.254 range.

We can deduce from this information that the source address carried out a scan of the FTP service on the addresses in question.

Site details:

nslookup: no match found
whois: no match found

Conclusion: This address has been seen to be scanning for systems that operate the FTP service. The scan was probably trying to identify the operating systems in use.

Recommendation: As above.

3.4. Priority 4 Detect (External RPC call)

The following table shows a summary of the probes made by various external systems against the RPC service on IP addresses within the University's IP address range. If the Remote Procedure Call portmapper service is running on a host then a client connecting to it will be given details of what programs have registered themselves with the portmapper. Subsequent connections can then be made to those services with possible access to data and/or processing power.

Date	Time	Src IP	Src P	Dst IP	Dst P

01-Jul	09:05:21	164.164.87.134	1606	MY.NET.132.0	111	From
01-Jul	09:05:23	164.164.87.134	2017	MY.NET.132.240	111	To
01-Jul	09:05:27	164.164.87.134	2070	MY.NET.133.3	111	From
01-Jul	09:05:34	164.164.87.134	2601	MY.NET.133.246	111	To
01-Jul	09:05:31	164.164.87.134	2645	MY.NET.134.35	111	From
01-Jul	09:05:41	164.164.87.134	3159	MY.NET.134.231	111	To
01-Jul	09:05:39	164.164.87.134	3199	MY.NET.135.15	111	From
01-Jul	09:05:44	164.164.87.134	3632	MY.NET.135.253	111	To
01-Jul	09:05:51	164.164.87.134	4193	MY.NET.137.1	111	From
01-Jul	09:06:02	164.164.87.134	4776	MY.NET.137.253	111	To
02-Jul	01:02:41	170.211.172.90	4720	MY.NET.132.175	111	From
02-Jul	01:02:41	170.211.172.90	4790	MY.NET.132.245	111	To
02-Jul	01:02:41	170.211.172.90	4800	MY.NET.133.0	111	From
02-Jul	01:02:41	170.211.172.90	1081	MY.NET.133.253	111	To
02-Jul	01:02:41	170.211.172.90	1093	MY.NET.134.9	111	From
02-Jul	01:02:41	170.211.172.90	1184	MY.NET.134.100	111	To
02-Jul	01:02:42	170.211.172.90	1369	MY.NET.135.29	111	From
02-Jul	01:02:44	170.211.172.90	1591	MY.NET.135.251	111	To
02-Jul	01:02:44	170.211.172.90	1851	MY.NET.137.0	111	From
02-Jul	01:02:44	170.211.172.90	1925	MY.NET.137.74	111	To
02-Jul	09:08:18	199.84.54.32	111	MY.NET.132.1	111	From
02-Jul	09:08:20	199.84.54.32	111	MY.NET.132.253	111	To
02-Jul	09:08:20	199.84.54.32	111	MY.NET.133.17	111	From
02-Jul	09:08:23	199.84.54.32	111	MY.NET.133.254	111	To
02-Jul	09:08:23	199.84.54.32	111	MY.NET.134.3	111	From
02-Jul	09:08:25	199.84.54.32	111	MY.NET.134.253	111	To
02-Jul	09:08:25	199.84.54.32	111	MY.NET.135.1	111	From
02-Jul	09:08:28	199.84.54.32	111	MY.NET.135.250	111	To
02-Jul	09:08:31	199.84.54.32	111	MY.NET.137.18	111	From
02-Jul	09:08:33	199.84.54.32	111	MY.NET.137.236	111	To
04-Jul	00:01:03	204.117.207.245	3826	MY.NET.133.58	111	From
04-Jul	00:01:03	204.117.207.245	3914	MY.NET.133.146	111	To
04-Jul	00:01:04	204.117.207.245	4119	MY.NET.134.96	111	From
04-Jul	00:01:04	204.117.207.245	4190	MY.NET.134.167	111	To
04-Jul	00:01:02	204.117.207.245	4361	MY.NET.135.83	111	From
04-Jul	00:01:02	204.117.207.245	4437	MY.NET.135.159	111	To
04-Jul	00:01:07	204.117.207.245	4823	MY.NET.137.35	111	From
04-Jul	00:01:07	204.117.207.245	4898	MY.NET.137.110	111	To
01-Jul	08:46:36	211.23.6.234	1448	MY.NET.132.28	111	From
01-Jul	08:46:47	211.23.6.234	2011	MY.NET.132.246	111	To
01-Jul	08:46:44	211.23.6.234	2098	MY.NET.133.73	111	From
01-Jul	08:46:53	211.23.6.234	2616	MY.NET.133.254	111	To
01-Jul	08:46:53	211.23.6.234	2618	MY.NET.134.1	111	From
01-Jul	08:46:59	211.23.6.234	2986	MY.NET.134.191	111	To
01-Jul	08:46:57	211.23.6.234	3229	MY.NET.135.25	111	From
01-Jul	08:47:07	211.23.6.234	3642	MY.NET.135.246	111	To
01-Jul	08:47:10	211.23.6.234	4247	MY.NET.137.1	111	From
01-Jul	08:47:19	211.23.6.234	4605	MY.NET.137.176	111	To

We can see that source addresses 164.164.87.134, 170.211.172.90, 199.84.54.32, 204.117.207.245 and 211.23.6.234 have scanned a comprehensive range of destination addresses from MY.NET.132.0 to MY.NET.137.255.

A SANS paper by David Reece at http://www.sans.org/newlook/resources/IDFAQ/blocking.htm (Reece) provides more insight into this problem.

The only one of the above IP addresses that could be resolved was:
Site details:

nslookup: no match found
whois: Server Name: NS.URETHANEEXPERTS.COM
IP Address: 204.117.207.245
Registrar: REGISTER.COM, INC.
Whois Server: whois.register.com

Conclusion: The observed scan was [part of] a network mapping scan to establish which addresses are in use in the address ranges and possibly what operating system[s] the scanned hosts are running.

Recommendation: If inbound connections are not required to TCP RPC, block them a t the external firewall.

3.4.1. Correlations

The paper at http://www.cert.org/advisories/CA -2000-17.html gives details of port 111 exploits.

Candidate vulnerability CAN -2000-0666 on the CVE database a t www.mitre.org also refers to this issue.

3.5. Priority 5 Detect (connect to 515 from outside)

Data from the logs being reported to destination port 515 (IANA port list name: 'printer', IANA description: 'spooler', protocols TCP and UDP) was summarised and analysed. The data showed that MY.NET addresses had been scanned in a similar way to those reported in 3.4 above.

Generating a link count for all inbound packets from 165.132.31.137 showed that each destination system rec eived either one or two packets. Allowing for the IDS to have dropped packets it is assumed that each destination system received *approximately* 2 packets. These were thought to be TCP SYNs and a check of the Snort scans files corroborated the suspicion.

Date	Time	Src IP	SrcP	Dst IP	DstP
03-Jul	05:37:56	255.255.255.255	31337	MY.NET.135.58	515

The packet shown above stood out from all the others that had targeted port 515. IP address 255.255.255.255 is known as the 'limited broadcast address' (Stevens, p. 171) and is not normally used as a *source* address. Port 31337 is associated with the Back Orifice exploit.

Conclusion: The observed scan was [part of] a network mapping scan to establish which addresses are in use in the address ranges and possibly wh at operating system[s] the scanned hosts are running. None of the systems responded to the scan.

Recommendation: If inbound connections are not required to the printer service, block them at the external firewall.

3.5.1. Correlations

Roderick Campbell observed this detect as his fifth highest occurrence in his GCIA paper that can be found at www.sans.org/y2k/practical/Roderick_Campbell_GCIA.doc .

The SANS alert at http://www.sans.org/newlook/alerts/port515.htm gives further insight into this type of probe.

4. 'Top Ten Talkers'

The top talkers are those source IP addresses who are reported upon most often by IDSs (i.e. who is allegedly *causing* the alerts, beware that source IP addresses *may* be spoofed).

The list of 'top talkers' generated by the 'top_talk.ksh' UNIX shell script (see below) from the selected Snort log files are:

Rank	IP Address	Occurrences
1	MY.NET.160.114	67101
2	211.207.15.190	30297
3	66.68.62.229	23506
4	205.188.233.121	15074
5	205.188.233.153	14936
6	148.223.228.15	13158
7	61.222.34.170	12136
8	205.188.244.249	9594
9	205.188.246.121	8459
10	MY.NET.217.10	8225

The details of the top five external tal kers are given below.

4.1. Top Talker No 1 (211.207.15.190)

The following command was used to analyse which Snort files gave details of packets involving this IP address:

```
grep -c ,211.207.15.190, scans.txt alert.txt oos.txt
```

The command searches for the patter n ",211.207.15.190, " in the three listed files and the '-c' flag causes the command to report the number of times the pattern was found in each file. The output of the command was:

```
scans.txt:30156    alert.txt:141      oos.txt:0
```

All the data from this IP address was found to be targeted to TCP port 21 (FTP control) on various addresses in the MY.NET.X.Y range. None of the Snort log files included evidence of any replies.

Site details:

nslookup: no match found
whois: no match found

4.2. Top Talker No 2 (66.68.62.229)

The above grep command was used again and gave the following results:

```
scans.txt:24671    alert.txt:6  oos.txt:1
```

Analysing the data to and from this IP address shows that all the data (all 24671 packets) were sent between this address and MY.NET.219.42. 23 500 of the packets were targeted at MY.NET.219.42 and the remaining 1171 packets were observed from MY.NET.219.42 to 66.68.62.229.

The range of ports in use was extremely wide although all the source ports used on the 66.68.62.229 system were in the range 61000-65095. These packets were used to scan ports from 1 to 15000 on the MY.NET.219.42 system. An example is shown here:

```
07/01,12:39:16,,MY.NET.219.42,4001,66.68.62.229,214,SYN **S*****
07/01,21:36:23,,66.68.62.229,61806,MY.NET.219.42,4001,SYN **S****    *
```

Site details:

nslookup: cs666862-229.austin.rr.com
whois: no match found

The site name 'cs666862-229.austin.rr.com' indicates that this site may be a dial-up service, dynamically allocated when required. It is unlikely that analysis of this IP address *after* the event will provide genuine information - the IP address has probably been re-allocated to another user.

4.3. Top Talker No 3 (205.188.233.121)

The above grep command was used again and gave the following results:

```
scans.txt:15059    alert.txt:15       oos.txt:0
```

All the data from this IP address was found to be targeted to UDP port 6970 on various addresses in the MY.NET.X.Y range. None of the Snort log files included evidence of any replies.

UDP port 6970 is used for receiving 'RealAudio' data. The '`nslookup`' command returns information that the site in question's name is: `g2lb4.spinner.com`, looking at http://www.spinner.com we find that this site is an 'Internet Radio Station'. The observed data appears to be systems running applications such as QuickTime or RealPlayer to receive audio (e.g. news feeds or music) over the Internet.

N.B. A trojan program known as 'Gate Crasher' listens on TCP port 6970, the activity seen in this example was all UDP and therefore not suspected to be Gate Crasher activity.

Site details:
nslookup: g2lb4.spinner.com
whois: no match found

4.4. Top Talker No 4 (205.188.233.153)

The above grep command was used again and gave the following results:

```
scans.txt:14921    alert.txt:15       oos.txt:0
```

This IP address' DNS name is '`g2lb5.spinner.com`' and is a sister site to system identified as Top Talker No. 3.

Site details:
nslookup: g2lb5.spinner.com
whois: no match found

4.5. Top Talker No 5 (148.223.228.15)

The above grep command was used again and gave the following results:

```
scans.txt:13110    alert.txt:48       oos.txt:0
```

Data from this IP address was found to be targeted to many UDP and TCP ports on various addresses in the MY.NET.X.Y range. **Some of the Snort log files included evidence of replies.**
Some examples of inbound packets are:

```
07/01 00:56:44  148.223.228.15 45255 MY.NET.253.43 25 SYN 21S***** RESERVEDBITS
07/01 01:05:47  148.223.228.15 32927 MY.NET.70.97 65176 NOACK 21SFRP*U RESERVEDBITS
07/01 01:06:00  148.223.228.15 4978 MY.NET.1.6 563 SYN 21S***** RESERV    EDBITS
```

These packets have been crafted, they are not generated by operating systems' network stacks. They show signs of malicious activity.

Site details:
nslookup: du-148-223-228-15.prodigy.net.mx
whois: no match found

Looking up prodigy.net.mx revealed:
Site details:
whois: Server Name: PRODIGY.NET.MX
IP Address: 148.235.168.60
Registrar: REGISTER.COM, INC.
Whois Server: whois.register.com

5. Analysis of Out Of Specification (OOS) Data

The name 'Out Of Specification' refers to data that does not conform to the defined ways of communicating using TCP/IP. The table below show the number of OOS packets detected for the top ten sources of OOS data, with their DNS names and whois information.

IP Address	Detects	DNS Name	Whois information
211.180.236.194	557	Not found	Name: NS.CWDESIGN.CO.KR Registrar: YESNIC CO. LTD. Whois Server: whois.yesnic.com
210.77.146.33	390	Not found	Name: NS2.YOULE.NET Registrar:BULKREGISTER.COM,INC. Whois Server: whois.bulkregister.com
199.183.24.194	175	vger.kernel.org	Name: NS.VGER.KERNEL.ORG Registrar: NETWORK SOLUTIONS, INC.
24.66.152.186	108	h24-66-152-186.gv.shawcable.net	Not found
216.5.180.10 193.226.113.248	41 31	Not found 248.valahia.ro	Not found Not found
209.150.103.212	18	Realityfailure.org	Name: DNS.REALITYFAILU RE.ORG Registrar: TUCOWS, INC. Whois Server: whois.opensrs.net
64.152.176.4	12	64-152-176-4-rev-l3.inyc.com	Not found
24.169.190.158	11	syr-24-169-190-158.twcny.rr.com	Not found
192.117.120.140	11	firegate.savan.com	Not found

Various of these addresse s were chosen for further analysis, the results of which are given below.

5.1. OOS Data: 211.180.236.194

All OOS data seen related to this address was from TCP port 111 (portmapper) to port 111 at various MY.NET.X.Y addresses. Some packets are shown below:

```
07/03,13:21:11,493913,211.180.236.194,111,MY.NET.132.1,111,TCP       TTL:25
TOS:0x0 ID:39426 **SF**** Seq: 0x695B4072 Ack: 0x44C85FBB Win: 0x404 00 00
00 00 00 00

07/03,13:21:11,529981,211.180.236.194,111,MY.NET.132.3,111,TCP       TTL:25
TOS:0x0 ID:39426 **SF**** Seq:  0x695B4072 Ack: 0x44C85FBB Win: 0x404 00 00
00 00 00 00

07/03,13:21:11,572401,211.180.236.194,111,MY.NET.132.5,111,TCP       TTL:25
TOS:0x0 ID:39426 **SF**** Seq: 0x695B4072 Ack: 0x44C85FBB Win: 0x404 00 00
00 00 00 00
```

Various characteristics can be seen in t his data:

1) The IP ID field (value:29426) doesn't change between packets;
2) The TCP flags SYN and FIN are both set, this is not a valid flag combination;
3) The TCP sequence number (value:0x695B4072) doesn't change between packets;
4) The TCP acknowledgement number (value:0x44C85FBB) doesn't change between packets;

Many more packets were seen like these, all destined to different destination IP addresses. We can conclude from this that this was a SYN/FIN scan against MY.NET.X.Y. This is corroborated by entries in Snort's scan files.

5.2. OOS Data: 210.77.146.33

All OOS data seen related to this address was directed to TCP port 80 (http) at two MY.NET.X.Y addresses. Some packets are shown below:

```
07/01,15:39:49,026156,210.77.146.33,34575,MY.NET.253.114,80,TCP      TTL:46
TOS:0x0 ID:29838 DF 21S***** Seq: 0x64C06770 Ack: 0x0 Win: 0x16D0 TCP
Options => MSS: 1460 SackOK TS: 4789246 0 EOL EOL EOL EOL

07/01,15:39:50,133797,210.77.146.33,34584,MY.NET.253.114,80,TCP      TTL:46
TOS:0x0 ID:36186 DF 21S***** Seq: 0x65689936 Ack: 0x0 Win:    0x16D0 TCP
Options => MSS: 1460 SackOK TS: 4789356 0 EOL EOL EOL EOL

07/01,15:39:51,396430,210.77.146.33,34608,MY.NET.253.114,80,TCP      TTL:46
TOS:0x0 ID:4826 DF 21S***** Seq: 0x654D0630 Ack: 0x0 Win: 0x16D0 TCP
Options => MSS: 1460 SackOK TS: 4789482 0 EOL   EOL EOL EOL
```

Of the 390 packets detected, 370 were directed at MY.NET.253.114 and the remaining 20 packets at MY.NET.100.165. 156 packets were targeted at MY.NET.253.114 on Jul 01 between 15:39:49 and 15:43:03. The packets have TCP reserved bits '2' and '1' and the SYN bit set, this is not a valid flag combination. The IP Identification field changes from 29838 to 36186 to 4826 over a period of two seconds. This is not normal behaviour and is another sign of crafted packets.

5.3. OOS Data: 24.169.190.158

The following are some of the packets observed from this IP address:

```
07/03,16:49:27,752393,24.169.190.158,2953,MY.NET.70.66,6346,TCP   TTL:108   TOS:0x0
ID:41555 DF 21S*R*** Seq: 0xD0188 Ack: 0x28F668F Win: 0x5010 ...

07/03,16:50:10,149447,24.169.190.158,0,MY.N  ET.70.66,2953,TCP    TTL:108    TOS:0x0
ID:25175 DF **SF*P*U Seq: 0x18CA0188 Ack: 0x8AAA6690 Win: 0x5010 ...

07/03,16:51:00,006281,24.169.190.158,0,MY.NET.70.66,2953,TCP    TTL:108    TOS:0x0
ID:29022 DF **SF**A* Seq: 0x18CA0189 Ack: 0x8BF66691 Win: 0x5010 TCP Option   s => NOP
NOP TS: 80020482 2864587741

07/03,16:51:04,401103,24.169.190.158,0,MY.NET.70.66,2953,TCP    TTL:108    TOS:0x0
ID:58974 DF 21*FRP** Seq: 0x18CA0189 Ack: 0x9A1C6691 Win: 0x5010 TCP Options =>
MSS: 1460 NOP WS: 1 Opt 17 (24): D155 0E55 0000 0000 0000 000   0 0000 0000 0000 0000
0000 EOL EOL EOL EOL EOL EOL EOL

07/03,16:53:01,313036,24.169.190.158,2953,MY.NET.70.66,6346,TCP   TTL:108   TOS:0x0
ID:40045 DF 21**RP** Seq: 0x18C97CE Ack: 0xED6695 Win: 0x5010 TCP Options => NOP
NOP TS: 1021564102 1772319503
```

```
07/03,16:53:09,629219,24.169.190.158,2953,MY.NET.70.66,6346,TCP    TTL:108    TOS:0x0
ID:53614 DF 21**RP** Seq: 0x18CDF34 Ack: 0xD6696 Win: 0x5010 ...
```

A check of all ports used to/from this address was performed by the following script call:

```
$ ./links.ksh 24.169.1 90.158 <oos.txt

24.169.190.158 2953 to MY.NET.70.66 6346 6
24.169.190.158 0 to MY.NET.70.66 2953 5
```

Ports TCP 6346 and UDP 6347 are commonly used by Gnutella software.

It is unusual to see a source side port (2953) suddenly be used as a source port for a return packet, especially when the destination port of that packet is 0, this may be a programming error.

The number of TCP options shown in the fourth packet leads to speculation that the packet may be intended to cause a buffer overrun.

5.4. OOS Data: 192.117.120.140

The following are some of the packets observed from this IP address:

```
07/03,05:14:54,309606,192.117.120.140,62741,MY.NET.70.27,6347,TCP   TTL:43   TOS:0x0
ID:12300 DF 21S***** Seq: 0xB733EE4F Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS:  74767939 0 EOL EOL EOL EOL
07/03,05:40:49,895834,192.117.120.140,64079,MY.NET.228.74,6346,TCP  TTL:43  TOS:0x0
ID:55274 DF 21S***** Seq: 0x1935EA3D Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 74923479 0 EOL EOL EOL EOL
07/03,05:40:52,907907,192.  117.120.140,64079,MY.NET.228.74,6346,TCP  TTL:43  TOS:0x0
ID:55275 DF 21S***** Seq: 0x1935EA3D Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 74923779 0 EOL EOL EOL EOL
07/03,05:40:58,967539,192.117.120.140,64079,MY.NET.228.74,6346,TCP  TTL:43  TOS:0    x0
ID:55276 DF 21S***** Seq: 0x1935EA3D Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 74924379 0 EOL EOL EOL EOL
07/03,07:31:32,627584,192.117.120.140,63711,MY.NET.70.27,6347,TCP   TTL:43   TOS:0x0
ID:20162 DF 21S***** Seq: 0xBCBEAF04 Ack: 0x0 Win:    0x16D0 TCP Options => MSS: 1460
SackOK TS: 385478 0 EOL EOL EOL EOL
07/03,07:44:48,950258,192.117.120.140,64231,MY.NET.228.74,6346,TCP  TTL:43  TOS:0x0
ID:26458 DF 21S***** Seq: 0xEE107F95 Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 465103 0 EOL  EOL EOL EOL
07/03,07:44:51,982874,192.117.120.140,64231,MY.NET.228.74,6346,TCP  TTL:43  TOS:0x0
ID:26459 DF 21S***** Seq: 0xEE107F95 Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 465403 0 EOL EOL EOL EOL
07/03,07:44:58,041895,192.117.120.140,6423  1,MY.NET.228.74,6346,TCP  TTL:43  TOS:0x0
ID:26460 DF 21S***** Seq: 0xEE107F95 Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 466003 0 EOL EOL EOL EOL
07/03,08:19:51,248526,192.117.120.140,61646,MY.NET.201.74,6346,TCP  TTL:43  TOS:0x0
ID:13156 DF 21S ***** Seq: 0x724C7E8F Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 675307 0 EOL EOL EOL EOL
07/03,08:19:54,291975,192.117.120.140,61646,MY.NET.201.74,6346,TCP  TTL:43  TOS:0x0
ID:13157 DF 21S***** Seq: 0x724C7E8F Ack: 0x0 Win: 0x16D0 TCP Options    => MSS: 1460
SackOK TS: 675607 0 EOL EOL EOL EOL
07/03,08:20:00,243730,192.117.120.140,61646,MY.NET.201.74,6346,TCP  TTL:43  TOS:0x0
ID:13158 DF 21S***** Seq: 0x724C7E8F Ack: 0x0 Win: 0x16D0 TCP Options => MSS: 1460
SackOK TS: 676207 0 EOL EOL EOL EOL
```

All the above packets show an unusual use of the TCP flags field, the bits '2' and '1' are not normally used.

The two packets targeted to MY.NET.70.27 are both to TCP port 6347. The other packets are to port 6346 (often used by Gnutella).

The two 'bursts' of three packets sent to MY.NET.228.74 and the one 'burst' of three packets sent to MY.NET.201.74 show delays of 2 -3 and then 5-7 seconds almost as if they were TCP re-transmissions (but they shouldn't change their IP ID if that were the case).

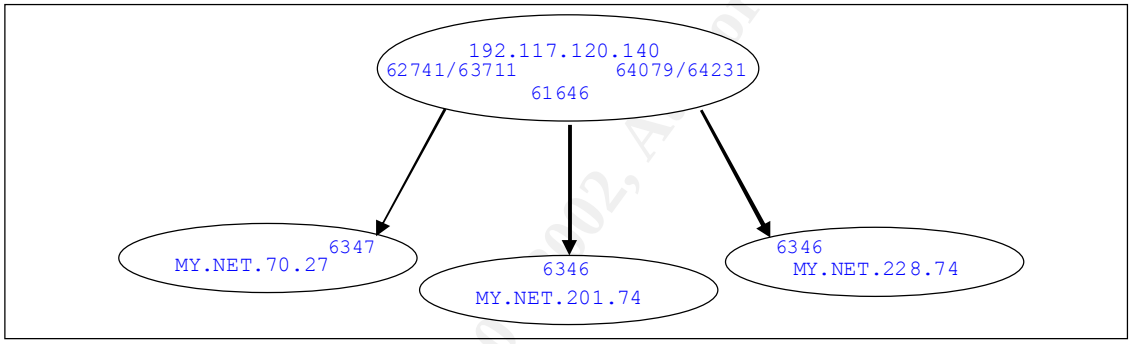
This seems to be an attempt to elicit a response from the targeted systems.

A check of all OOS data to/from this address was performed by the following script call:

```
$ ./links.ksh 192.117.120.140 <oos.txt | ./linkg.ksh 1 10

192.117.120.140 61646  ->->      3  ->-> 6346  MY.NET.201.74
                64079  ->->      3  ->-> 6346  MY.NET.228.74
                64231  ->->      3  ->-> 6346
                62741  --->      1  ---> 6347  MY.NET.70.27
                63711  --->      1  ---> 6347
```

The following link graph represen ts this data in a graphical form:



6. Concerns about internal systems

The list of top talkers lists two internal systems: MY.NET.160.114 and MY.NET.217.10.

MY.NET.160.114 was observed performing many portscans against external systems. T his activity may be illegal. The identity of MY.NET.160.114 should be established and steps taken to ensure that its activity is legitimate.

MY.NET.217.10 was observed performing many portscans against external systems. This activity may be illegal. Th e identity of MY.NET.217.10 should be established and steps taken to ensure that its activity is legitimate.

MY.NET.217.154 TCP port 1214 has been observed communicating with 199.4.19.2. TCP port 1214 is used by 'Morpheus' software (a file sharing progra m). The Morpheus software is known to have a security problem where it can used to gain access to a system's private data. MY.NET.217.154 should be checked to ensure it is not 'leaking' University data.

Other internal systems are causing concern. These should be overcome if the systems as a whole are scanned for vulnerabilities/trojans etc. and any such situations eradicated. The issue will then be how to keep them clean!

7. Defensive Recommendations

It is recommended that the University develop a network architecture that allows the bulk of its systems to be behind firewalls. The ruleset on the firewalls for *inbound* data should be established on a 'need to have' basis: only allow access where users can demonstrate that they need to have it. Systems prov iding data to the outside world can have access explicitly provided to them by the ruleset. Outbound access through the firewall should be fairly open as the University presumably wishes to encourage learning and investigation.

Any Internet facing firewa lls should be configured to deny any packets inbound or outbound that have source or destination ports within the range defined as Private IP Addresses in RFC 1918.

It is suggested that a baseline of standards be devised and adopted for all operating syst ems in use. This baseline should be regularly updated to take advantage of various patches to overcome vulnerabilities and other operating systems shortfalls as they become available.

To protect those systems which are used to explore out on the Internet (and those internally into which users install unknown software) it is suggested that regular checks are made to ensure that no rogue software is operating and that the security baseline is intact. Tools such as CyberCop (Commercial) and Nessus (freely a vailable) can be used for this task.

8. Analytical Process

The chosen data was copied to a UNIX system where it was subjected to various custom written scripts.

Reference was made to UNIX shell scripts in PJ Goodwin's GCIA practical at www.sans.org/y2k/practical/PJ_Goodwin_GCIA.doc (Goodwin) although the scripts used there could not be used here due to the format of the comma separated files used in this analysis being different from those that PJ Goodwin used.

The scripts used here process the raw Snort data into a consistent comma separated format:

Date,Time,Fraction_of_Second,Src_IP,Src_Port,Dst_IP,Dst_Port,Other

where Other varies according to the type of data being processed.

Where a data field is not present a null field is generated. Beware that there may be commas in the *final* field, typically where the data is the ASCII representation of the data (payload) part of a packet in the OOS data.

The scripts used for this analysis ar e available from the author's web site in zipped tar format or tar format for those who may wish to use them for their own purposes.

The scripts and other files are also included here. Beware that some lines get wrapped by the software used to view this document (e.g. MS Word). You may have to reduce the pitch of the font to see lines correctly.

The results of the UNIX sh ell scripts were copied to a MS Windows NT system where MS Excel was used to manipulate the data (primarily to sort the data using various key fields).

The Snort alert files were processed with a Korn shell script named 'alert.ksh' (which uses a sed pattern file named 'alert.sed') to produce a file called alert.txt by the following command:

```
cat alert.01070[1 -5].txt | alert.ksh >alert.txt
```

The Snort scans files were processed with a Korn shell script named 'scans.ksh' (which uses an sed pattern file named 'scans.sed') to produce a file called scans.txt by the following command:

```
cat scans.01070[1 -5].txt | scans.ksh > scans.txt
```

The Snort OOS files were processed with a Korn shell script named 'oos.ksh' (which uses an sed pattern file named 'oos.sed') to pro duce a file called oos.txt by the following command:

```
cat oos.01070[1 -5].txt | oos.ksh > oos.txt
```

The output from the above scripts was processed to generate 'top talkers' list with a Korn shell script named 'top_talk.ksh':

```
cat alert.txt scans.txt oos.txt  | top_talk.ksh > tt.txt
```

The output from the above scripts was processed to generate 'top destinations' list with a Korn shell script named 'top_dest.ksh':

```
cat alert.txt scans.txt oos.txt | top_dest.ksh > td.txt
```

Selected IP addresses were extracted from selections of the above files and used to generate lists of associated ports and other addresses/ports to which they have been observed to communicate (i.e. to be 'linked') , for example:

```
links.ksh 65.8.220.176 <alert.txt >links.alert.65.8.220.176.txt
```

Which generates data of the form:

```
IP_Address Port Direction IP_Address Port Link_Count
```

Sample:

```
65.8.220.176 4660 to MY.NET.10.4 27374 4
65.8.220.176 4636 to MY.NET.9.235 27374 4
65.8.220.176 4626 to MY.NET.9.225 27374 4
65.8.220.176 4754 to MY.NET.10.102  27374 3
65.8.220.176 4752 to MY.NET.10.100 27374 3
65.8.220.176 4748 to MY.NET.10.96 27374 3
65.8.220.176 4739 fm MY.NET.10.86 27374 3
65.8.220.176 4708 fm MY.NET.10.54 27374 3
…
```

This data was then processed to generate a crude 'link graph' with the fol lowing command:

```
linkg.ksh 1 4 <links.alert.65.8.220.176.txt >linkg.a.65.8.220.176.txt
```

Which generates data of the form:

```
IP_Address Port Pointer Link_Count Pointer Port IP_Address
```

Sample:

```
65.8.220.176     4656  ->->      2 ->-> 27374 MY.NET.10.0
                 4666  ->->      3 ->-> 27374 MY.NET.10.10
                 4752  ->->      3 ->-> 27374 MY.NET.10.100
                 4754  ->->      3 ->-> 27374 MY.NET.10.102
                 4756  ->->      2 ->-> 27374 MY.NET.10.104
                 4668  --->      1 ---> 27374 MY.NET.10.13
                 4670 < -<-      2 <-<- 27374 MY.NET.10.15
                 4670  ->->      2 ->-> 27374
                 4672  --->      1 ---> 27374 MY.NET.10.17
                 4658  ->->      3 ->-> 27374 MY.NET.10.2
```

Note 1: Where a line does not show an IP address on the left or the right it is because the address is the same as on the previous line.

Note 2: The 'boldness' of the pointers is intended to indicate the magnitude of the link, the arguments given to the linkg.k sh command ('1' and '4' in this example) tell the script what link counts to consider as low magnitude (shown as ' --->') and what link counts to consider high magnitude ('>>>>'). Magnitudes between these values are shown as ' ->->'.

List of References

Albanna, Z. et al. IANA Guidelines for Ipv4 Multicast Address Assignments (RFC 3171) . Juniper Networks, et al. 2001

Goodwin, PJ. GCIA practical. URL: www.sans.org/y2k/practical/PJ_Goodwin _GCIA.doc , 2000

Maughan, D. et al. Internet Security Association and Key Management Protocol (RFC 2408) . National Security Agency, et al, 1998

Murray, Bill. et al. "Protection of Home/SOHO Systems with Persistent Connections and IP Addresses." SANS News Bites. Vol. 3 Num. 37 (2000)

Reece, David. Is blocking port 111 sufficient to protect your systems from RPC attacks? . SANS Institute, URL: http://www.sans.org/newlook/resources/IDFAQ/b locking.htm (12 Sep. 2001)

Rekhter, Y., et al. Address Allocation for Private Internets (RFC 1918) . Cisco Systems et al, 1996

St. Johns, M. Identification Protocol (RFC 1413) . US DoD, 1993

Stevens, W. Richard. TCP/IP Illustrated, Volume 1 . Reading: Addison Wesley Longman Inc, 1994.

ICMP Type Numbers . 29 Jun. 2001. URL: http://www.iana.org/assignments/icmp -parameters (16 Aug. 2001)

Troll, R. Automatically Choosing an IP address in an Ad -Hoc Ipv4 Network. Oct 1998, URL: http://www.ietf.org/proceedings/98dec/I -D/draft-ietf-dhc-ipv4-autoconfig-01.txt (Sep 2001)

Last Updated: June 25th, 2018

Upcoming Training

SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
Security Operations Summit & Training 2018	New Orleans, LA	Jul 30, 2018 - Aug 06, 2018	Live Event
San Antonio 2018 - SEC503: Intrusion Detection In-Depth	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
Community SANS Columbia SEC503	Columbia, MD	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, Japan	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, Netherlands	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS London September 2018	London, United Kingdom	Sep 17, 2018 - Sep 22, 2018	Live Event
SANS Network Security 2018	Las Vegas, NV	Sep 23, 2018 - Sep 30, 2018	Live Event
SANS Brussels October 2018	Brussels, Belgium	Oct 08, 2018 - Oct 13, 2018	Live Event
SANS Northern VA Fall- Tysons 2018	Tysons, VA	Oct 13, 2018 - Oct 20, 2018	Live Event
SANS Denver 2018	Denver, CO	Oct 15, 2018 - Oct 20, 2018	Live Event
SANS October Singapore 2018	Singapore, Singapore	Oct 15, 2018 - Oct 27, 2018	Live Event
Mentor Session - SEC503	Ankara, Turkey	Oct 31, 2018 - Dec 19, 2018	Mentor
Mentor Session - SEC503	Ballston, VA	Nov 01, 2018 - Dec 06, 2018	Mentor
SANS Dallas Fall 2018	Dallas, TX	Nov 05, 2018 - Nov 10, 2018	Live Event
San Diego Fall 2018 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Nov 12, 2018 - Nov 17, 2018	vLive
SANS San Diego Fall 2018	San Diego, CA	Nov 12, 2018 - Nov 17, 2018	Live Event
SANS Stockholm 2018	Stockholm, Sweden	Nov 26, 2018 - Dec 01, 2018	Live Event
Tactical Detection & Data Analytics Summit & Training 2018	Scottsdale, AZ	Dec 04, 2018 - Dec 11, 2018	Live Event
SANS Cyber Defense Initiative 2018	Washington, DC	Dec 11, 2018 - Dec 18, 2018	Live Event
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced