Global Information Assurance Certification Paper

GCIA Practical Assignment

Version 3.5

Campus NIDS in focus

Rusma Mulyadi
09/20/2004

Abstract

This paper starts with an IDS design that utilizes the ***EtherChannel*** loadbalancing algorithm of Cisco Catalyst 6500. It is then followed by a discussion of three network detects: ***Misc. Tiny Fragments***, ***Large ICMP*** and ***My Doom M/O***. The paper concludes with an '***Analyze This***' section that discuss the findings and recommendations for MY.EDU network based on my analysis on a 5-days log files.

Table of Content

Part 1 - IDS Design

The network described in this paper is a university network with approximately 37,000 enrolled students and 1,700 faculty members. The whole network is spread over more than 500 buildings and cover almost 1000 different departments. Similar to any other university, providing a secure computing environment is always a challenge, especially due to its decentralized and open nature. In addition, budget is another limiting factor.

This paper, especially the on-campus IDS architecture, is mainly designed to allow effective use of IDS sensors, especially in an environment where the distribution network layer consists of multiple switches of which the network traffic varies quite significantly. Thus, placing an IDS sensor on each distribution switch is an inefficient and quite expensive solution. This design will try to accommodate these issues.

Product Specifications

Border IDS[1]
The Cisco Intrusion Detection Service Module 2 is used as the border IDS, with the specification below:

- ✓ Cisco IDSM-2 Ver.4.1
- ✓ Platform: WS-SVC-IDSM2-BUN
- ✓ Pentium P3 1.13 GHz on main board with 232MHz IXP 32 bit StrongARM
- ✓ 100G hard drive (20G used)
- ✓ 2G RAM, 4G Event Storage, 64MB Flash
- ✓ Performance: 600 Mbps with 450-byte packets; up to 4000 TCP connections per second; up to 500,000 concurrent connections

The IDSM-2 is installed on a Cisco Catalyst 6500 together with a Firewall Service Module (FWSM). This combination allows the IDSM-2 to send shunning/blocking requests to the Firewall whenever certain signatures are triggered.
Since there is only 1 IDSM-2 is implemented on the network, it is still recommended to use the Command Line Interface (CLI) for configuration management. In addition, the IDSM-2 also includes Cisco IDS Device Manager (IDM), a web browser interface to the configuration management. Note: Although CLI is still my preferred management option, there are certain tasks related to shunning/blocking through the FWSM currently can be performed properly though the IDM/ IDS MC. IDS MC (IDS Management Center) is another

[1] http://www.cisco.com/en/US/customer/products/hw/modules/ps2706/products_data_sheet09186a00801e55dd.html

management option that needs to be purchased separately. Unless, we are adding more IDSM-2 to the network in the future, IDS MC is not necessary.

On-campus IDS[2]

As for the on-campus IDSs, the Sourcefire Network Sensors and Real-time Network Awareness (RNA) sensors are used. The specifications are as follow:

2 units of NS3000 Ver3.2

- ✓ 2G RAM
- ✓ Speed: 1 Gigabit
- ✓ Disk space: 30GB

2 units of RA3000 Ver3.2

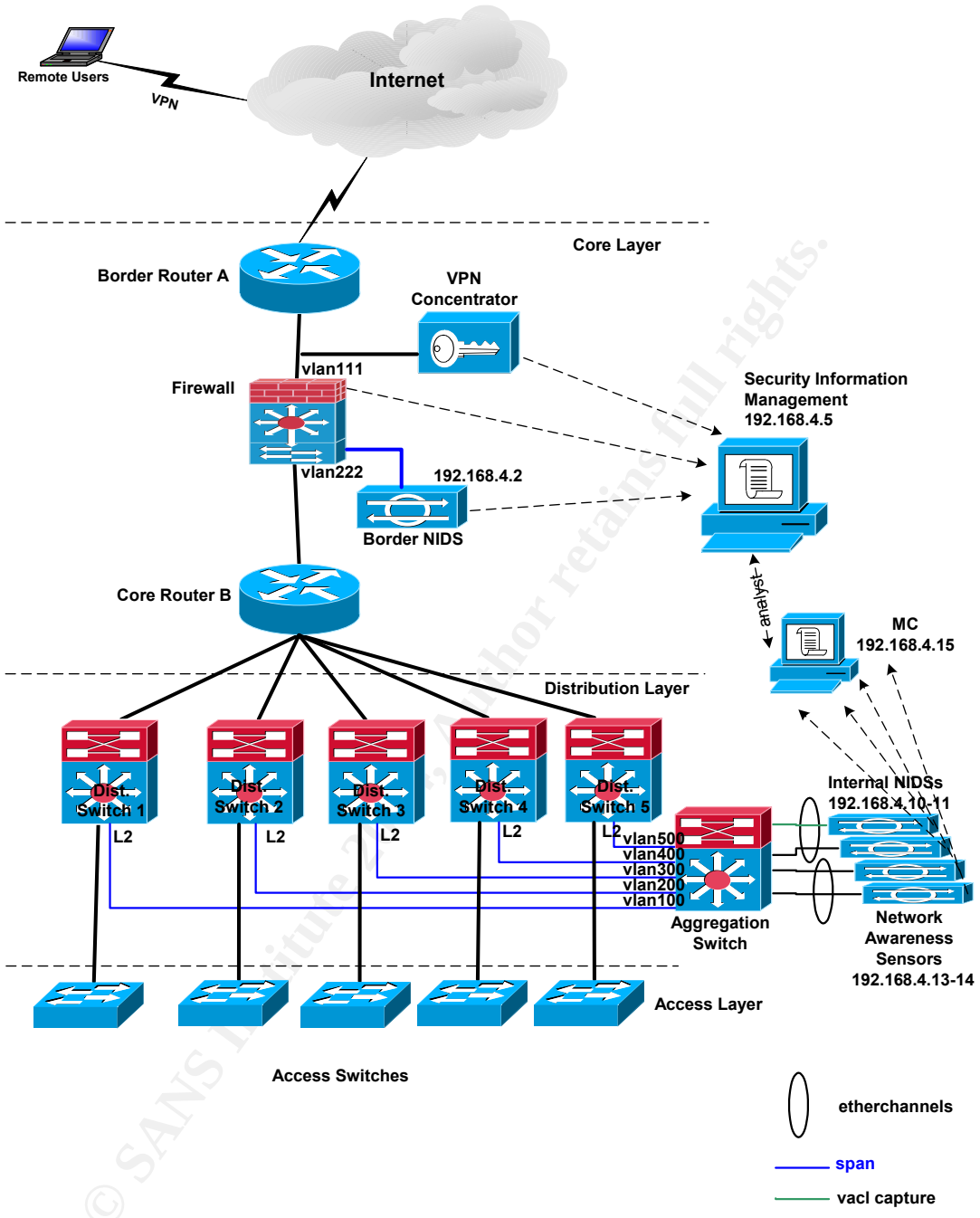
- ✓ Host licenses: 4 x 8192 = 32,768 hosts
- ✓ 2GB RAM
- ✓ Disk space: 30GB

A management console unit (MC3000) with the following specification is used to manage these 4 sensors:

- ✓ 4GB RAM
- ✓ Disk space: 300 GB

[2] www.sourcefire.com

Network Diagram



IDS Management

Border IDS

The IDSM-2 configuration management is performed through:

1. Command Line Interface/CLI, accessible via:

a. Direct SSH connections from specific subnets to the management interface, i.e. ***192.168.4.2*** in this case.
b. Console port of the Cisco Catalyst 6500 (IOS) using the ***session slot*** command

2. IDS Device Manager, accessible through HTTPS connections from specific subnets.

Signature and software updates are still downloaded manually from Cisco's website and then pulled from the sensors via the ***upgrade*** command from the sensor's CLI.

IDSM-2 utilized a communications protocol called ***Remote Data Exchange Protocol*** (RDEP) that allow alerts being pulled from the sensor instead of being pushed by the sensor. In addition, the data being exchanged is in XML format and comply to IDIOM (Intrusion Detection Interchange and Operations Messages), which is "*a data format standard that defines the event messages that are reported by the IDS as well as the operational messages that are used to configure and control the IDS*"[3].

An in-house RDEP client is developed to pull data from the sensor and parse the XML log files into a MySQL database. The database schema is also designed in-house based on the Cisco's RDEP documentation.

On-campus IDS

The configuration management for these Network Sensor and RNA sensors are performed through the Management Console (MC) web interface (MC's management IP: ***192.168.4.15***) that is accessible via ***HTTPS*** from specific subnets. Furthermore, all system and events monitoring and administration are also performed via the MC, including downloads of rules updates from Sourcefire website.

Monitored Traffic

Border IDS

The border IDS is configured to capture traffic off the inside interface of the border firewall, which is vlan222 in this case.

```
# enable idsm module to capture on vlan222
intrusion-detection module 4 data-port 1 capture
intrusion-detection module 4 data-port 1 capture allowed-vlan 222

# create an access map to capture traffic that match the gcia-vacl
```

[3] Cisco IDIOM Documentation

```
vlan access-map gcia-vacl-map 10
 match ip address gcia-vacl
 action forward capture
!
# apply the access map to vlan 222
vlan filter gcia-vacl-map vlan-list 222

# define the vacl
ip access-list extended gcia-vacl
 permit ip any any
```

On-campus IDS

In order to monitor all on-campus traffic without having to put a network sensor on each distribution switch, a layer-2 aggregation switch is used. All traffic are spanned from each distribution switches to different vlans on the aggregation switch. Using the EtherChannel algorithm on the aggregation switch, all traffics are loadbalanced and forwarded to the sensors. VACL capture is used to aggregate and forward the traffic to the EtherChannels. We need two EtherChannel in this particular case because all captured traffic need to be forward to both the network sensors and RNA sensors.

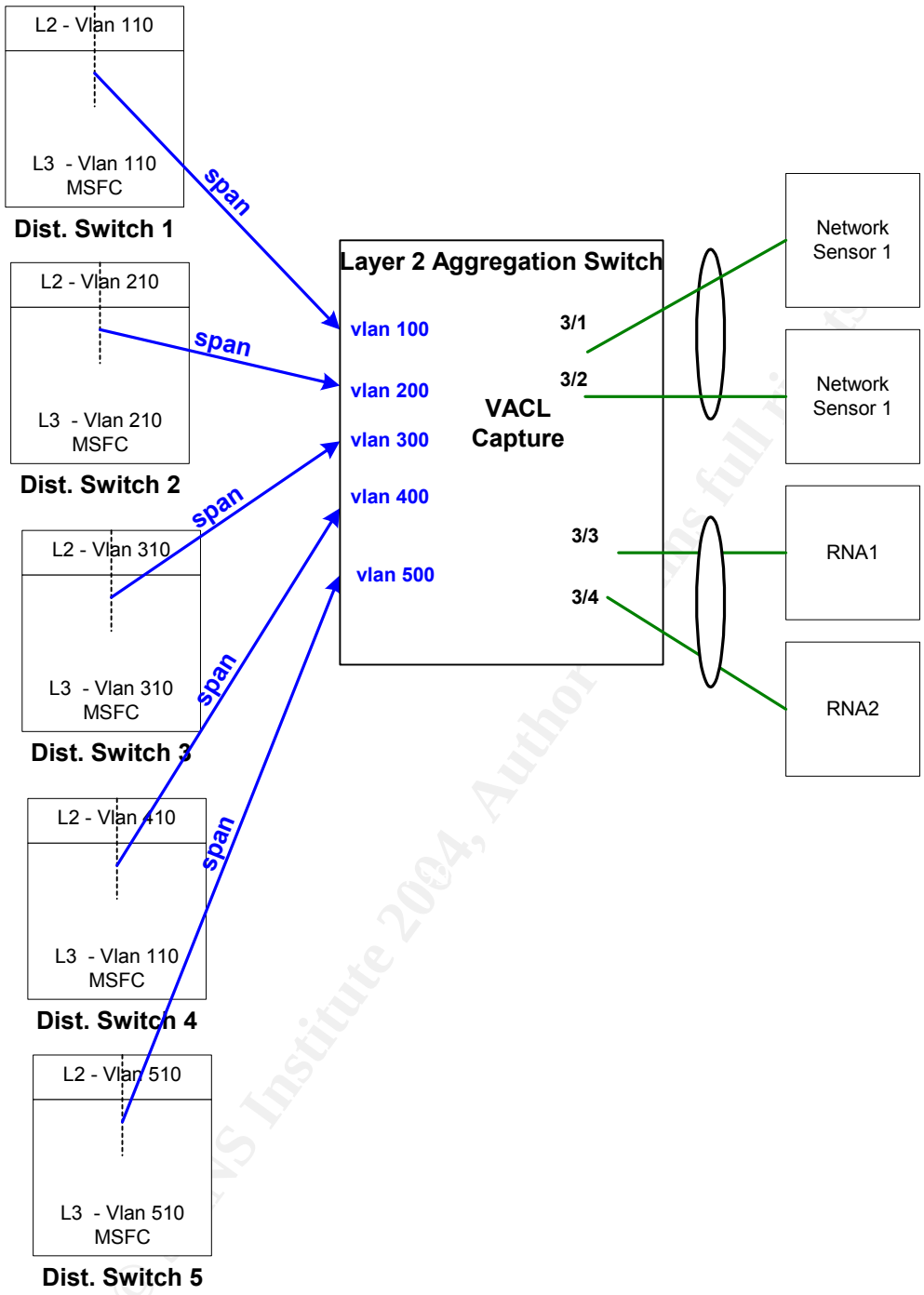


Figure 1 On campus IDS traffic flow

```
Sample configuration on the aggregation switch:
# Enable trunking an the capture ports: 3/1-4 in this case
set trunk 3/1  on dot1q
set trunk 3/2  on dot1q
set trunk 3/3  on dot1q
set trunk 3/4  on dot1q
```

```
# specify the capture ports that will receive traffic from the VACL
set security acl capture-ports 3/1-2
set security acl capture-ports 3/3-4

# configure the VLAN ACL:
set security acl ip gcia permit arp
set security acl ip gcia deny udp any any eq 1985
set security acl ip gcia deny pim any any
set security acl ip gcia deny eigrp any any
set security acl ip gcia deny ip 224.0.0.0 15.255.255.255 any
set security acl ip gcia deny ip any 224.0.0.0 15.255.255.255
set security acl ip gcia deny ip host 224.0.0.1 any
set security acl ip gcia deny ip any host 224.0.0.1
set security acl ip gcia permit ip any any capture

# commit the VACL
commit security acl all

# enable spantree bpdu filter to avoid conflicts
set spantree bpdu-filter 2/1-6 enable

# map the VACL to desired vlans
set security acl map gcia 100-500

# configure the EtherChannel between the aggregation switch and the
sensors
set port channel 3/1-2 mode on
set port channel 3/3-4 mode on
```

Alert Management and Reporting

As showed in the network diagram, all alerts from firewall, border IDS, VPN concentrator are stored into different databases on a database server (192.168.4.5). Currently, there is an on-site hot backup of the database server. Furthermore, the alerts from on-campus IDSs and RNA sensors, they are handled by the Sourcefire MC.

In the next future, there are needs to look and test both open source and commercial tools that will perform the security information management such OSSIM and Protego.

We do produce daily reports from both the border and on-campus IDSs. Depending on the state of the network, we also produce adhoc reporting (hourly to 4-hourly) to identify certain worm propagation activities.

Reference:

http://www.cisco.com/en/US/customer/products/hw/modules/ps2706/products_data_sheet09186a00801e55dd.html
www.sourcefire.com
Cisco IDIOM Documentation

Part2 – Network Detect

Detect 1: Misc. Tiny Fragments

Source of Trace:

The source of this detect is obtained from http://isc.sans.org/logs/Raw/2002.6.1 - 2002.6.18 and dated from 06/30/02 to 07/18/02.

Several key points to keep in mind throughout this network detect:[4]

- ✓ Only packets that violate the Snort rule set appear in these log files
- ✓ The logs have been sanitized:
  a. The IP addresses of the protected network space have been "*munged*"
  b. The checksums have also been modified
  c. Certain keywords have been modified
  d. All ICMP, DNS, SMTP and web traffic has been removed

These log raw log files are then combined into a single file called 2002.6.all using ***pcapmerge***[5], downloaded as part of ***tcpreplay*** package[6].

```
# pcapmerge -o 2002.6.all 2002.6.*

-o <file>: used to specify the name of the combined output file.
```

The next section discusses the network layout of this detect based on the ***tcpdump***[7] command and its sample output below.

```
# tcpdump -ennqr <filename>

-e    prints the link-level header, containing physical/MAC addresses
-nn   disables the host name and port translations
-q    quick output with less protocol information
-r    reads in the merged raw log file

17:31:16.304488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 60: 211.152.3.40.80 >
46.5.15.174.80: tcp 0
```

The fields of the above ***tcpdump*** output format – delimited by spaces/blanks – that will be analyzed further in this section are:

```
Field#            Description
   2                 Source MAC Address
   3                 Destination MAC Address
   5                 Source IP Address and Port
   7                 Destination IP Address and Port
```

[4] http://isc.sans.org/logs/Raw/README
[5] http://tcpreplay.sourceforge.net/pcapmerge.html
[6] http://www.sourceforge.net/projects/tcpreplay/
[7] http://tcpdump.org/tcpdump_man.html

To obtain the unique source MAC addresses, the *second* field of the sample output above is uniquely sorted. The resulted source MAC addresses are printed together with their number of occurrences in the entire log:

```
# tcpdump -ennqr 2002.6.all | awk '{print $2}' | sort | uniq -c
  32479 0:0:c:4:b2:33
   6406 0:3:e3:d9:26:c0
```

The third field of the output file identifies the destination MAC addresses and is printed below:

```
# tcpdump -ennqr 2002.6.all | awk '{print $3}' | sort | uniq -c
   6406 0:0:c:4:b2:33
  32479 0:3:e3:d9:26:c0
```

In addition to the fact that the counts of both MAC addresses as the source and destination addresses are exactly reversed, both MAC addresses are also registered to Cisco Systems according to the information shown below from the IEEE OUI Registration database[8]. Therefore, these logs are generated by an intrusion detection system (sniffer) that sits between two Cisco devices.

```
00-00-0C    (hex)              CISCO SYSTEMS, INC.
00000C      (base 16)          CISCO SYSTEMS, INC.
                               170 WEST TASMAN DRIVE
                               SAN JOSE CA 95134-1706

00-03-E3    (hex)              Cisco Systems, Inc.
0003E3      (base 16)          Cisco Systems, Inc.
                               170 West Tasman Dr.
                               San Jose CA 95134
                               UNITED STATES
```

To further understand the location of these devices within the network:

1. The distinct source IP addresses (field #5) coming from **0:0:c:4:b2:33**

```
# tcpdump -ennqr 2002.6.all 'ether src 0:0:c:4:b2:33' | awk '{print
  $5}' | awk -F \. {'print $1 "." $2 "." $3 "." $4'} | sort |
  uniq -c | sort -rn
  32434 46.5.180.250
     45 46.5.180.133
```

2. The distinct destination IP addresses (field #7) coming from **0:0:c:4:b2:33**

```
# tcpdump -ennqr 2002.6.all 'ether src 0:0:c:4:b2:33' | awk '{print
  $7}' | awk -F \. {'print $1 "." $2 "." $3 "." $4'} | sort |
```

[8] http://standards.ieee.org/regauth/oui/oui.txt

```
uniq -c | sort -rn
16700 64.154.80.51
 6339 66.128.224.70
  686 64.154.80.50
  522 208.45.133.13
  <snip>
    1 12.216.161.118
    1 12.150.245.163
    1 12.111.145.134
```

At this point, it seems that **0:0:c:4:b2:33** is a Cisco device that sits in front of a Class C internal network (46.5.180.0/24) at this point.

3. The distinct source IP addresses (field #5) coming from **0:3:e3:d9:26:c0**

```
# tcpdump -ennqr 2002.6.all 'ether src 0:3:e3:d9:26:c0' | awk
  '{print $5}' | awk -F \. {'print $1 "." $2 "." $3 "." $4'} | sort
  | uniq -c | sort -rn
  683 255.255.255.255
  297 66.220.44.31
  180 203.122.47.137
  169 62.153.209.202
   <snip>
    1 12.150.54.250
    1 12.107.51.109
    1 12.105.86.5
```

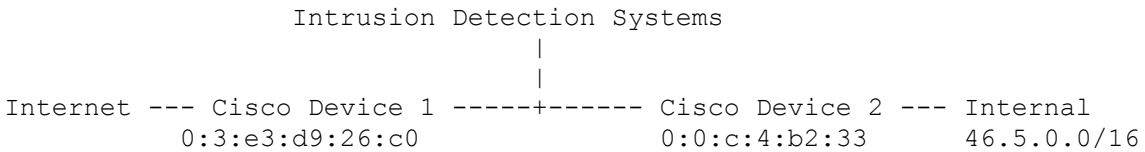
Based on the variety of IP addresses sourced from **0:3:e3:d9:26:c0**, we can conclude that it is most likely a '*border*' router that is located between the Internet and the network in this particular trace.

4. The distinct destination IP addresses (field #7) coming from **0:3:e3:d9:26:c0**

```
# tcpdump -ennqr 2002.6.all 'ether src 0:3:e3:d9:26:c0' | awk
  '{print $7}' | awk -F \. {'print $1 "." $2 "." $3 "." $4'} | sort
  | uniq -c | sort -rn | more
  1714 46.5.180.250
  908 46.5.180.133
  381 46.5.80.149
   36 46.5.218.182
   28 46.5.130.100
   26 46.5.180.135
  <snip>
    1 46.5.0.16
    1 46.5.0.139
    1 46.5.0.130
    1 46.5.0.10
```

From this last output, we discover that the internal network is much bigger than a Class C network. Instead, it looks more like a Class B network (46.5.0.0/16).

The network layout of this trace can then be summarized as:



5. The distinct destination ports coming from **0:0:c:4:b2:33**

```
### Total unique destination ports coming from 0:0:c:4:b2:33
# tcpdump -ennqr 2002.6.all 'ether src 0:0:c:4:b2:33' | awk '{print
  $7}' | awk -F \. {'print $5'} | sort | uniq -c | sort -rn | wc -l
  248

### List of unique destination ports coming from 0:0:c:4:b2:33
# tcpdump -ennqr 2002.6.all 'ether src 0:0:c:4:b2:33' | awk '{print
  $7}' | awk -F \. {'print $5'} | sort | uniq -c | sort -rn
  29487 80:
    851 6347:
    552 1863:
    410 6348:
     95 5555:
    <snip>
      1 10508:
      1 10450:
      1 10222:
      1 100:
```

The existence of 248 unique destination ports of the outgoing traffic in the entire log file indicates a loose egress filtering. However, it seems that there are filters for the well-known Microsoft Windows ports (135, 137-139, and 445) because none of them are found in these 248 destination ports. The top 4 outgoing traffic corresponds to http (port 80), gnutella (port 6347 and 6348), MSN messenger (1863) and Napster (port 5555).[9] This is assuming that these ports are used by their normal applications.

6. The distinct destination ports coming from **0:3:e3:d9:26:c0**

```
### Total unique destination ports coming from 0:3:e3:d9:26:c0
# tcpdump -ennqr 2002.6.all 'ether src 0:3:e3:d9:26:c0' | awk
  '{print $7}' | awk -F \. {'print $5'} | sort | uniq -c | sort -rn
  | wc -l
  717

### List of unique destination ports coming from 0:3:e3:d9:26:c0
# tcpdump -ennqr 2002.6.all 'ether src 0:3:e3:d9:26:c0' | awk
  '{print $7}' | awk -F \. {'print $5'} | sort | uniq -c | sort -rn
   1704 80:
    705 21:
    683 515:
    591 53:
```

[9] http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html

```
 416 6346:
<snip>
   4 137:
<snip>
   1 61002:
   1 6000:
   1 41992:
   1 40195:
   1 3514:
```

The existence of 717 unique destination ports in the incoming traffic that includes a Windows NetBIOS port 137 in the entire log file also indicates a loose ingress filtering. Although it seems that the egress filtering includes a block for well-known Windows ports, it does not appear to be the case with regard to the ingress filtering. The top 3 incoming traffic corresponds to http (port 80), ftp (port 21), printer (port 515), domain name (port 53), and gnutella (port 6346).[9] Again, this is assuming that these ports are used by their normal applications.

The limited ingress and egress filtering applied on this network is an indication of a very loose network security policy implementation and it fits the profile of typical **.edu** network.

Detect was generated by:

This detect is generated using Snort Version 2.2.0 with all the rules enabled. The latest rules are The **snort.conf** is modified to include as the home network. The following command is used:

```
# snort -r 2002.6.all  -c ../snort/etc/snort.conf -A full
  -l alert612ALL/ -dey -k none

-r <filename> read and process a tcpdump file
-c <cfgfile>  use rules specified in the configuration file
-A <mode>     alert mode (fast, full, console or none)
-l <log dir>  specify the log directory
-d            print the application layer information
-e            display the 2nd layer header info
-y            include year in timestamp of alert and log files
-k <mode>     checksum mode (all, noip, notcp, noudp, noicmp, none)
```

SnortSnarf v021111.1[10] is then used to facilitate the alert analysis process.

```
# ./snortsnarf -d alert612view/ alert612ALL/alert

-d <dir>      specify the output directory
```

[10] http://www.snort.org/dl/contrib/data_analysis/snortsnarf/

This detect will focus on the ***Misc Tiny Fragments*** signature alerts as summarized in Figure 2 below.

MISC Tiny Fragments	5 sources	1 destinations
Priority: 2	Classification: Potentially Bad Traffic	
[sid:522]		

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
64.26.170.95	1	1	1	1
64.105.26.118	1	1	1	1
64.105.26.164	1	1	1	1
217.83.201.131	1	1	1	1
80.136.103.85	1	1	1	1

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
46.5.80.149	5	22	5	19

Figure 2 Misc Tiny Fragments Alerts Summary

An alert on this signature is generated when a fragment packet (with ***More Fragment – M*** flag set) with a payload size (***dsize***) of less than 25 bytes is received from the external network. Below is the corresponding Snort rule[11].

```
alert ip $EXTERaNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments";
dsize:< 25; fragbits:M; classtype:bad-unknown; sid:522; rev:2;)
```

The 5 alerts of interest are:

```
[**] [1:522:2] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
```

[11] http://www.snort.org/snort-db/?sid=522

```
07/10/02-01:06:04.854488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
64.26.170.95 -> 46.5.80.149 TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1FCB Frag Size: 0x0014
[**] [1:522:2] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/10/02-20:41:23.524488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
80.136.103.85 -> 46.5.80.149 TCP TTL:241 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1E17 Frag Size: 0x0014
[**] [1:522:2] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/10/02-22:48:25.114488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
64.105.26.118 -> 46.5.80.149 TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1ED3 Frag Size: 0x0014
[**] [1:522:2] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/11/02-16:38:45.424488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
64.105.26.164 -> 46.5.80.149 TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1ED3 Frag Size: 0x0014
[**] [1:522:2] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/15/02-21:26:14.184488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
217.83.201.131 -> 46.5.80.149 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1EB2 Frag Size: 0x0014
```

Using the last alert above as a sample, a Snort alert format contains:

```
[**] [1:522:2] MISC Tiny Fragments [**]
```

→ **[**] [*Generator ID*: *Signature ID*: *Revision Number*] *Signature Message* [**]**

```
[Classification: Potentially Bad Traffic] [Priority: 2]
```

→ **[Classification: *Classification Type's Short Name*] [Priority: *Priority level*]**

```
07/15/02-21:26:14.184488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
```

→ ***Timestamp Source MAC Address -> Destination MAC Address***
→ **type: *encapsulation protocol (0x800 = IP)***
→ **len: *length of the frame (0x3C = 60)***

```
217.83.201.131 -> 46.5.80.149 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x1EB2 Frag Size: 0x0014
```

→ ***Source IP Address -> Destination IP Address***
→ ***Protocol ID***
→ **TTL: *time to live***
→ **TOS: *type of service***
→ **ID: *IP / Fragment ID***
→ **DgmLen: *total length of datagram***
→ **IP Flag: *IP flags (reserved bit| Don't Fragment | More Fragment)***
→ **Frag Offset: *fragment offset***
→ **Frag Size: *fragment size***

To confirm these Snort generated alerts, the log file is passed through ***tcpdump*** with ***bpf filters*** to include packets with:

1. ***More Fragment*** flag set (`ip[6] & 0x20 != 0`)
2. ***total payload*** less than ***25 bytes*** (`ip[3]-((ip[0] & 0x0f)*4) < 25`).

```
# tcpdump -Xxvnns 1514  -r 2002.6.all '(ip[6] & 0x20 != 0) and (ip[6] &
  0x80 ==0) and (ip[3]-((ip[0] & 0x0f)*4) < 25)'
```

Packet 1: ip[6] == 0x6f --> fragbits:DM

```
17:50:45.374488 64.244.110.164 > 46.5.212.116: tcp (frag 0:20@30904+)
(ttl 237, len 40, bad cksum b1ae!)
0x0000   4500 0028 0000 6f17 ed06 b1ae 40f4 6ea4       E..(..o.....@.n.
0x0010   2e05 d474 8105 0050 0400 ff56 0400 ff56       ...t...P...V...V
0x0020   5004 0000 7ad3 0000 0000 0000 0000            P...z.........
```

Packet 2: ip[6] == 0x3e --> fragbits:M

```
21:26:14.184488 217.83.201.131 > 46.5.80.149: tcp (frag 0:20@62864+)
(ttl 242, len 40, bad cksum 70b1!)
0x0000   4500 0028 0000 3eb2 f206 70b1 d953 c983       E..(..>...p..S..
0x0010   2e05 5095 8389 18ca 0e11 7516 0e11 7516       ..P.......u...u.
0x0020   5004 0000 f3d1 0000 0000 0000 0000            P.............
```

Packet 3: ip[6] == 0x3e --> fragbits:M

```
20:41:23.524488 80.136.103.85 > 46.5.80.149: tcp (frag 0:20@61624+)
(ttl 241, len 40, bad cksum 5d46!)
0x0000   4500 0028 0000 3e17 f106 5d46 5088 6755       E..(..>...]FP.gU
0x0010   2e05 5095 8206 18ca 0935 7f24 0935 7f24       ..P......5.$.5.$
0x0020   5004 0000 d5ea 0000 0000 0000 0000            P.............
```

Packet 4: ip[6] == 0x3e --> fragbits:M

```
22:48:25.114488 64.105.26.118 > 46.5.80.149: tcp (frag 0:20@63128+)
(ttl 237, len 40, bad cksum bd88!)
0x0000   4500 0028 0000 3ed3 ed06 bd88 4069 1a76       E..(..>.....@i.v
0x0010   2e05 5095 8039 18ca 9d55 7d9e 9d55 7d9e       ..P..9...U}..U}.
0x0020   5004 0000 0f81 0000 0000 0000 0000            P.............
```

Packet 5: ip[6] == 0x3e --> fragbits:M

```
16:38:45.424488 64.105.26.164 > 46.5.80.149: tcp (frag 0:20@63128+)
(ttl 237, len 40, bad cksum bd5a!)
0x0000   4500 0028 0000 3ed3 ed06 bd5a 4069 1aa4       E..(..>....Z@i..
0x0010   2e05 5095 82ff 18ca a129 6dae a129 6dae       ..P......)m..)m.
0x0020   5004 0000 24c5 0000 0000 0000 0000            P...$.........
```

Packet 6: ip[6] == 0x3f --> fragbits:M

```
01:06:04.854488 64.26.170.95 > 46.5.80.149: tcp (frag 0:20@65112+) (ttl
237, len 40, bad cksum 2cf6!)
0x0000   4500 0028 0000 3fcb ed06 2cf6 401a aa5f       E..(..?...,.@.._
0x0010   2e05 5095 8047 18ca 4107 12d2 4107 12d2       ..P..G..A...A...
0x0020   5004 0000 0e0e 0000 0000 0000 0000            P.............
```

Packet 7: ip[6] == 0x64 --> fragbits:DM

```
08:00:17.944488 61.205.39.211 > 46.5.80.149: tcp (frag 0:20@9368+) (ttl
236, len 40, bad cksum 8e07!)
0x0000   4500 0028 0000 6493 ec06 8e07 3dcd 27d3       E..(..d.....=.'.
0x0010   2e05 5095 81d1 18ca 1c07 fa00 1c07 fa00       ..P.............
0x0020   5004 0000 0d00 0000 0000 0000 0000            P.............
```

The ***tcpdump*** command results in two additional packets, i.e. Packet 1 and 7. This is because these two packets also have the ***Don't Fragment*** flag set in addition to the ***More Fragment*** flag. The corresponding snort rule fires when

only the More Fragment is set. The ***bpf filter*** is then modified to fit this restriction (**`ip[6] & 0x60 == 0x20`**). Below is the modified ***tcpdump*** command and only packet 2 – 5 are returned this time.

```
# tcpdump -Xxvnns 1514 -r 2002.6.all '(ip[6] & 0x60 == 0x20) and
(ip[3]-((ip[0] & 0x0f)*4) < 25)'
```

Probability the source address was spoofed:

As will be further described in the next two sections, I believe that the packets in this trace are corrupted RST packets from real Gnutella traffic. Therefore, the possibility that the source addresses are spoofed is very small.

The ***TTL*** (***time to live***) values also appear to be consistent with the source IP classes, i.e. the one sourced from the same Class A (64.x.x.x) seems to have similar TTL values. Searching these 5 source addresses in ***Geektools***[12] returns three different Internet Service Provider companies, i.e.: Magma Communication/ Canada, Covad/ US, Deutsche Telekom AG.

Also, performing ***nslookup*** on these machines shows that they are alive.

```
Name:    pD953C983.dip.t-dialin.net
Address:  217.83.201.131

Name:    h-64-105-26-164.lsanca54.dynamic.covad.net
Address:  64.105.26.164

Name:    h-64-105-26-118.lsanca54.dynamic.covad.net
Address:  64.105.26.118

Name:    p50886755.dip0.t-ipconnect.de
Address:  80.136.103.85

Name:    ottawa-hs-64-26-170-95.d-ip.magma.ca
Address:  64.26.170.95
```

Thus, these 5 source addresses are mostly likely regular ***Gnutella servents***.

In addition, RST packets are usually sent either in responses to connection requests to non-existence services, to abort existing connections, or to OS fingerprint systems in the target network. In this particular detect, I tend to believe that the corrupted RST packets are generated to abort existing Gnutella connections, and therefore, the source addresses are not spoofed.

Description of attack:

Tiny fragments are often to bypass the intrusion detection systems and firewalls that fail to perform proper fragments reassembly. The common fragmentation

[12] http://www.geektools.com/whois.php

attacks include fragmentation overlap, fragmentation overwrite, and fragmentation time-outs.[13] Although most of the current intrusion detection systems (e.g. ***Snort***'s ***frag2*** preprocessor) and firewalls are capable of maintaining states and perform fragmentation reassembly, these devices are still susceptible to denial of service attacks while not configured properly. In addition, some systems may crash or severely disrupted when receiving a lot of malformed fragmented packets.[14]

In this particular ***tiny fragments*** detect, there are two possible explanations:

1. These packets are part of a larger tiny fragment attacks described above. The facts that the ***frag id*** and the ***data length*** are always ***0*** and ***20*** might indicate that these are crafted packets. However, there is not enough information available to support this conclusion, especially with only 1 fragment of the entire fragments train is available in the packet trace.
2. These are corrupted packets and based on further description in the next section, they are corrupted ***RST*** packets sent by a ***Gnutella servent*** to end a connection. I incline towards this second scenario.

Attack mechanism:

There are several interesting similarities among the packets in this detect:

1. The 4th and 5th bytes offset of IP header – ***IP/fragment ID*** = 0
2. The ***More Fragment*** flag – found in the 3 high-order bits of the 6th bytes offset of IP header – is always set. The 6th bytes offset of IP headers (***ip[6]***) of this detect are either ***0x3e*** or ***0x3f***. Using the IP header template:

```
Hex   => Decimal  => IP Flags | Fragment Offset (portion)
                     xDM
0x3e  => 62       => 001      | 11110
0x3f  => 63       => 001      | 11111

IP Flags:
x - reserved, set to 0; D - Don't Fragment; M - More Fragment
```

3. The 9th byte offset of the IP header – ***Protocol*** = ***0x06*** (i.e. ***TCP***)
4. The 2nd and 3rd bytes offset of the payload (since the MF flag is set in the IP header) = ***0x18ca*** or decimal ***6346***
5. The values of the 4th to 7th bytes offset of the payload is always the same as those of the 8th to 11th bytes offset
6. The 12th and 13th bytes offset of the payload = ***0x5004***

[13] http://www.securityfocus.com/infocus/1577
[14] http://www.securiteam.com/windowsntfocus/Patch_Available_for_the__IP_Fragment_Reassembly__Vulnerability.html

If I ignore the fact that these packets are fragments and consider the payload as TCP header instead of fragment payload, point 4 – 6 above can be translated into:

4. Destination port = 6346 (***Gnutella***)
5. Sequence number = ACK number
6. TCP header length = 20 bytes; TCP RST flag is set

```
Hex          => Offset    | Reserved   | Flags
                          |            | UAPRSF
0x5004       => 0101      | 0000 00    | 000100

TCP Flags:
U - Urgent; A - Ack; P - Push; R - Reset; S - Syn; F - Fin
```

To support my suspicion that these packets are corrupted ***Gnutella RST*** packets, I use ***tcpdump*** to filter all traffic sourced from / destined to 46.5.80.149 in the raw log file. (Notice that all of these alerts destined to the same ip address, i.e. 46.5.80.149)

```
# tcpdump -nnr 2002.6.all 'dst host 46.5.80.149' | wc
    399    3536   35076
# tcpdump -nnr 2002.6.all 'src host 46.5.80.149' | wc
      0       0       0
```

While none of the logged traffics sourced from 46.5.80.149, there are 399 packets destined to it. Among these 399 packets: [15]

- ✓ 380 destined to TCP port 6346 - registered to Gnutella.
- ✓ 18 fragmented packets – without destination ports
- ✓ 1 destined to TCP 3514 – registered to MUST Peer to Peer

```
# tcpdump -nnr 2002.6.all 'dst host 46.5.80.149' | awk '{print $4}'
  | awk -F \. {'print $5'} | sort | uniq -c |more
     18
      1 3514:
    380 6346:
```

In addition, it appears that there are no packets sourced from the five different source IP addresses in this detect other than the ones triggering these ***Misc Tiny Fragment*** alerts.

```
# tcpdump -nnr 2002.6.all 'host 64.26.170.95 or host 64.105.26.118
  or host 64.105.26.164 or 217.83.201.131 or 80.136.103.85' |wc
     5      35     340
```

At this point, I can conclude that 46.5.80.149 is a ***Gnutella servent***. Below is a sample of the packets:

[15] http://www.iana.org/assignments/port-numbers

```
18:21:45.164488 148.63.247.123.3536 > 46.5.80.149.6346: P
2391248906:2391249078(172) win 8192 (DF)
0x0000   4500 00d4 9dcf 4000 6e06 6c04 943f f77b       E.....@.n.l..?.{
0x0010   2e05 5095 0dd0 18ca 8e87 900a 0000 0000       ..P.............
0x0020   5e08 2000 48d3 0000 474e 5554 454c 4c41       ^...H...GNUTELLA
0x0030   2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573       .CONNECT/0.6..Us
0x0040   6572 2d41 6765 6e74 3a20 4265 6172 5368       er-Agent:.BearSh
0x0050   6172 6520 322e 362e 330d 0a4d 6163 6869       are.2.6.3..Machi
0x0060   6e65 3a20 312c 3133 2c31 3930 2c31 2c34       ne:.1,13,190,1,4
0x0070   3938 0d0a 506f 6e67 2d43 6163 6869 6e67       98..Pong-Caching
0x0080   3a20 302e 310d 0a48 6f70 732d 466c 6f77       :.0.1..Hops-Flow
0x0090   3a20 312e 300d 0a4c 6973 7465 6e2d 4950       :.1.0..Listen-IP
0x00a0   3a20 3134 382e 3633 2e32 3437 2e31 3233       :.148.63.247.123
0x00b0   3a36 3334 360d 0a52 656d 6f74 652d 4950       :6346..Remote-IP
0x00c0   3a20 3137 302e 3132 392e 3230 342e 3139       :.170.129.204.19
0x00d0   0d0a 0d0a                                     ....
```

Gnutella is a peer-to-peer protocol for distributed search and digital distribution / file sharing. Each participant is called a ***servent*** and acts as both a client and a server.[16] A description on the protocol can be found on the link below:

http://rfc-gnutella.sourceforge.net/developer/index.html

From my online research, I was not able to obtain much information regarding TCP RST packets to end Gnutella connections. I then decide to capture some live Gnutella (port 6346) traffic on our campus network to learn more on its behavior. From the captured traffic, I try to find RST packets with the same sequence (***tcp[4:4]***) and ACK numbers (***tcp[8:4]***).

```
### capture live Gnutella (port 6346) traffic
# tcpdump -i bond0 -xnns 1514 'port 6346'

### filter RST packets with sequence # = ACK #
# tcpdump -r gnu -xnns 1514 'tcp[13] = 0x04 and tcp[4:4] = tcp[8:4] and
dst port 6346'
```

I do find quite a lot of these packets. Below are some samples (checksums and source IPs are removed):

```
16:40:22.417450 x.x.x.x.4834 > 68.162.158.172.6346: R 4231098138:42
31098138(0) win 0
                        4500 0028 a541 0000 7f06 xxxx xxxx xxxx
                        44a2 9eac 12e2 18ca fc31 6f1a fc31 6f1a
                        5004 0000 f492 0000 0000 0000 0000
16:40:22.459180 x.x.x.x.4839 > 69.177.14.249.6346: R 4232152131:423
2152131(0) win 0
                        4500 0028 a545 0000 7f06 xxxx xxxx xxxx
                        45b1 0ef9 12e7 18ca fc41 8443 fc41 8443
                        5004 0000 58c0 0000 0000 0000 0000
16:40:22.622202 x.x.x.x.4837 > 24.51.185.190.6346: R 4231561837:423
1561837(0) win 0
```

[16] http://rfc-gnutella.sourceforge.net/developer/share/intro.html#Background

```
4500 0028 a551 0000 7f06 xxxx xxxx xxxx
1833 b9be 12e5 18ca fc38 826d fc38 826d
5004 0000 df38 0000 0000 0000 0000
```

Although RST packets are usually sent either as responses to connection requests to closed ports or as ways to abort existing connections based the RFC 793[17], I notice that Gnutella servents often send both FIN (normal way to terminate connections) and RST packets when closing a connection unless the remote servents respond immediately with FIN ACK packets. This duplication is particularly true when a *busy* message ("*There are too many active upload, and no space in the queues"*) is received from a remote servent. It seems that this is intended by design to avoid a lot of TCP half-close connections and to tear down the connections right away.

Since not all of these live captured RST packets have the same sequence and ACK numbers, I perform OS fingerprinting (using nmap) on two of these machines and they are identified as Windows systems as showed below.

```
Remote operating system guess: Windows Millennium Edition (Me), Win
2000, or Win XP
Remote OS guesses: Windows NT 5 Beta2 or Beta3, Windows Millennium
Edition (Me), Win 2000, or WinXP, MS Windows2000 Professional RC1/W2K
Advance Server Beta3
```

In addition, ***p0f***'s ***RST+ signatures***[18] describes that "*while the ACK value should be zeroed, it is not strictly against the RFC, and some systems either leak memory there or set it to the value of SEQ. The latter variant, with non-zero ACK, is particularly common on Windows"*.

Thus, if the packets are really corrupted RST Gnutella packets, I can be pretty confident that the source IPs are Windows machines.

The next question is how do these ***RST*** packets turn into ***fragments***? Comparing a sample of the real RST packets and the ones in this detect, there is an obvious different is on the 4th – 7th bytes offset of the IP header. It seems that the values of the first two bytes (4th and 5th – ***0x0000***) are somehow swapped with those of the last two (6th and 7th – ***0x3fcb***) in this detect. Since the 4th & 5th bytes offset of IP header are the ***IP/Fragment ID*** and the 6th & 7th bytes offset are ***IP Flags*** and ***Fragment Offset***, swapping them around can definitely turn a non-fragment packet into a fragment.

Real Gnutella RST packet

```
4500 0028 a551 0000 7f06 xxxx xxxx xxxx
1833 b9be 12e5 18ca fc38 826d fc38 826d
5004 0000 df38 0000 0000 0000 0000
```

[17] http://www.faqs.org/rfcs/rfc793.html
[18] http://www.stearns.org/p0f/p0fr.fp

Last packet in this detect (Packet 6)

```
4500 0028 0000 3fcb ed06 2cf6 401a aa5f
2e05 5095 8047 18ca 4107 12d2 4107 12d2
5004 0000 0e0e 0000 0000 0000 0000
```

To support this swapping theory, I notice that my earlier tcpdump filter on tiny fragment packets (less than 25 bytes payload and MF flag set) returned two additional packets that have both MF (More Fragment) and DF (Don't Fragment) flags set. These two flags identify 2 opposing traits of a packet and should not exist together in a packet when the RFC is followed. Thus, I am pretty sure that these two packets are also corrupted by the same technique.

Correlations:

There are several postings regarding '***MISC Tiny Fragments***' detect on the intrusions list:

- ✓ Richard Haynal analyzed a different traffic trace in www.incidents.org/logs/Raw/2002.9.22 and also concluded that the packets in his detect are corrupted. The original posting is available at: http://www.dshield.org/pipermail/intrusions/2003-April/007375.php

- ✓ Lesa Ludwig analyzed another traffic trace in http://www.incidents.org/logs/Raw/2002.10.11. Several categories of fragmentation attacks are presented as part of the analysis. The original posting is available at: http://www.dshield.org/pipermail/intrusions/2003-January/006463.php

Considering that the tiny fragments in this detect are corrupted packets, there are no related CVE entries.

Although I am reluctant to believe that this detect is part of a tiny fragmentation attack, below are several articles on the fragmentation attacks:

- ✓ IDS Evasion Techniques and Tactics, by Kevin Timm, May 7, 2002, http://www.securityfocus.com/infocus/1577

- ✓ Protection Against a Variant of the Tiny Fragment Attack, June 2001, http://rfc.net/rfc3128.html

- ✓ An Analysis of Fragmentation Attacks, by Jason Anderson, March 15, 2001, http://www.inet-sec.org/docs/DoS/fragma.html

- ✓ Cisco PIX and CBAC Fragmentation Attack, September 11, 1998, http://www.cisco.com/warp/public/770/nifrag.shtml

- ✓ Security Considerations for IP Fragment Filtering, October 1995, http://rfc.net/rfc1858.html

Only one of the five source IP addresses in this detect has an entry in ***myNetWatchman*** incident database[19], i.e. 217.83.201.131 (Incident ID: 112675479). However, this particular event is relatively new and related to Microsoft SMB/CIFS/Sasser/Agobot/Generic Bot, which is not relevant to this detect. No incident entry is found on ***dshield.org*** for these five source IPs.

Evidence of active targeting:

Should this detect be classified as a real attack, I can be relatively positive that there is an active targeting because all alerts in this detect are directed to the destination IP. However, this detect is not a real attack, but some corrupted packets. Thus, there is no evidence of active targeting in this particular detect.

Severity: -1
severity = (criticality + lethality) - (system countermeasures + network countermeasures)
= (2+1) – (2+2) = 3 – 4 = -1

Criticality: 2

Although there is no information regarding the criticality of this target system, the fact that traffic destined to it is mostly peer-to-peer related (as discussed earlier) lead me to conclude that it's an end-user workstation. Also, since this particular network fits the profile of an ***.edu*** network, an end-user workstation with Gnutella client program can be anyone's computer (student/faculty/staff). Although the criticality level may vary depending on the owner of the systems, I assign the criticality value of 2 here.

Lethality: 1

There is no real attack in this particular detect. I believe that these tiny fragments are corrupted Gnutella RST packets. Thus, the level of damage caused is very low.

System countermeasures: 2

There is no information regarding the system-level defenses on the target machine and I can only assume based on the fact that it is running a Gnutella client program. Since the security awareness level of average end users in an ***.edu*** environment is usually relatively low, I assign the system countermeasures a value of 2.

Network countermeasures: 2

[19] http://www.mynetwatchman.com/ListIncidentsbyIP.asp

Based on the information in the first section of this detect, I conclude that the existing network countermeasures seem to include loose egress & ingress filterings and an intrusion detection system. Although it seems that there are blocks for well-known Windows ports as part of the egress filtering, the same blocks do not appear to be applied in the ingress filtering. I therefore assign the network countermeasures a value of 2.

Defensive recommendation:

Although this detect does not include real fragmentation attacks, network–level defenses can still be put in place to prevent them and to further improve the current state of network security:

- ✓ Implement *stateful* firewall that is capable of maintaining *inter-fragment* state and dropping illegal and tiny fragments. If the size of the initial fragment is not large enough to fit all necessary header information, it should be dropped.[20] Any non-initial fragments should also be discarded unless the corresponding initial fragment has passed the firewall.[21] The amount of memory assigned to maintain the fragmentation state should also be limited to reduce the possibility of denial of service attacks against the firewall itself.

- ✓ Most of today's IDSs are capable of performing fragmentation reassembly, including Snort. Assuming Snort is used in this network, the ***frag2*** preprocessor need to be enabled.

- ✓ Applying more rigid ingress filtering by gradually moving from '***permit all, deny specifics***' to '***permit specifics, deny all***'. This might not be a easy option especially if this is an actual ***.edu*** network.

As for the host-level defenses:

- ✓ Fragmentation attacks may also cause certain un-patched systems to crash. Thus, it is required to keep each host up-to-date on its security patches.

- ✓ Assuming that the traffic categorization from the log file is a decent representation of the actual traffic on the network, peer-to-peer traffic is one of the most popular traffic coming into and leaving the network. As peer-to-peer programs usually come with spyware/ adware, the dangerous of these 'add-on' programs should be brought up to end users' attention. This is especially important when applying a more rigid ingress filtering is not a viable option.

Multiple choice test question:

[20] http://www.inet-sec.org/docs/DoS/fragma.html
[21] http://www.cisco.com/warp/public/770/nifrag.shtml

```
'tcp[13] = 0x04 and tcp[4:4] = tcp[8:4] and dst port 6346'
```

What kind of traffic does the above bpf filter look for?

A) TCP traffic with SYN and ACK flags set, the same source and destination ports, destination port = 6346
B) TCP traffic with RST flag set, the same source and destination ports, destination port = 6346
C) TCP traffic with SYN and ACK flags set, the same SEQ and ACK numbers, destination port = 6346
D) TCP traffic with RST flag set, the same SEQ and ACK numbers, destination port = 6346

Answer: D

This practical was posted twice to intrusions@incidents.org on August 31, 2004 and September 13, 2004 without any feedback responses.

Detect 2: Large ICMP

Source of Trace:

This trace is obtained from both the border and on-campus IDS of an ***.edu*** network as showed in Figure 3 below.

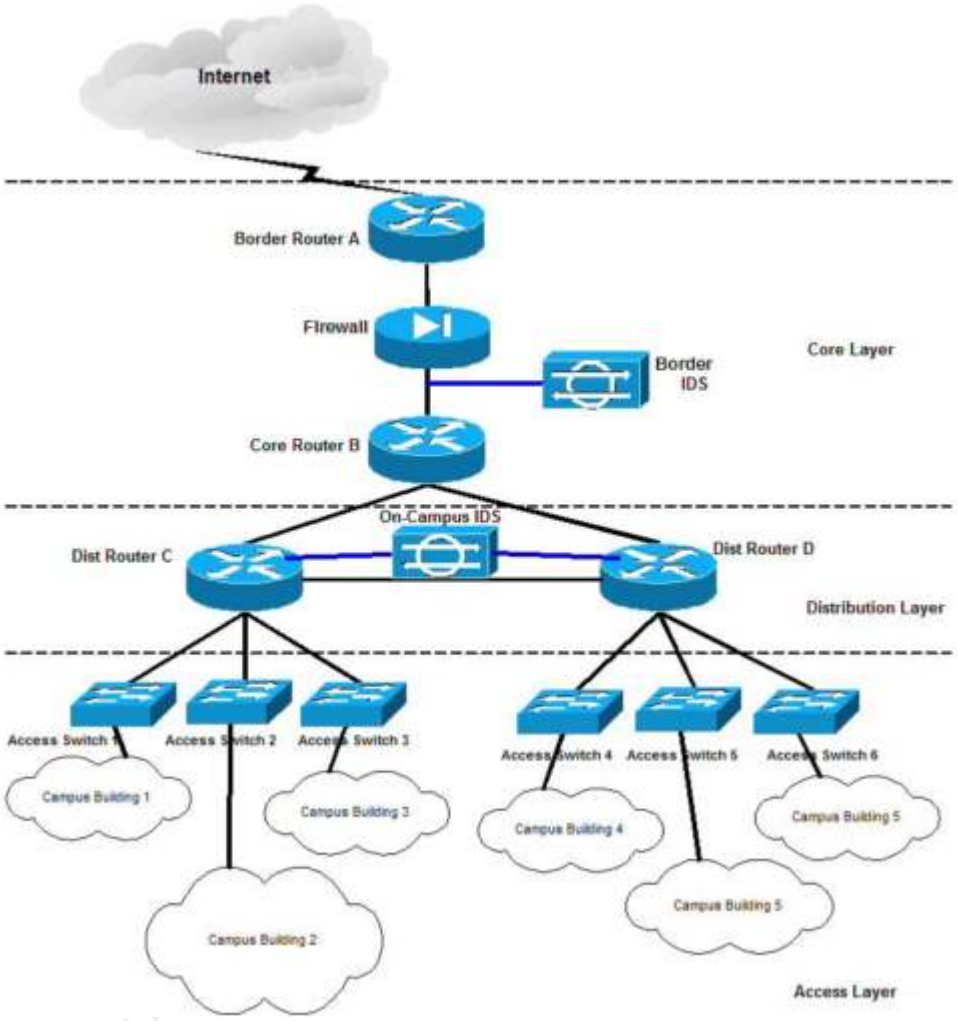


Figure 3. Network Diagram – Detect 2

As in most ***.edu*** network, the ingress and egress filtering at the border router and firewall are fairly loose. In this case, only certain popular backdoor and worm activity ports are blocked, including the NetBIOS ports. In addition, the border IDS is configured to issue temporal blocks requests to the border firewall when certain signature alerts are triggered.

Detect was generated by:

This detect is generated by both the border IDS (Cisco Secure Intrusion Detection System Version.4) and on-campus IDS (Snort 2.2.0). It first attracts my attention when it showed up in the border IDS daily top attacker/victim report. Below is an excerpt of this custom report, where ***X.X.X.X*** (an internal source address) is the attacker IP address with a total of 5903 daily alerts that consist of four different signatures.

Attacker IP	Hits	Hostname
X.X.X.X	5903	X.edu

Signature	Sub-Signature	Hits	Severity	Signature Description
2151	0	5867	0	Large ICMP
6901	0	12	0	NET FLOOD Icmp Reply
6903	0	12	0	NET FLOOD Icmp Any
6902	0	12	0	NET FLOOD Icmp Request

A sample of the raw event alerts, before being parsed into a database, is represented by the ***evAlert*** XML element below:

```
<evAlert eventId="1088064581975106010" severity="informational">
  <originator>
    <hostId>IDS-XXX</hostId>
    <appName>sensorApp</appName>
    <appInstanceId>1193</appInstanceId>
  </originator>
  <time offset="0" timeZone="UTC">1092066963081400000</time>
  <interfaceGroup>0</interfaceGroup>
  <vlan>XXX</vlan>
  <signature sigId="2151" subSigId="0" sigName="Large ICMP"
version="1.0"/>
  <participants>
    <attack>
      <attacker">
        <addr locality="IN">X.X.X.X</addr>
      </attacker>
      <victim>
        <addr locality="OUT">64.91.255.158</addr>
      </victim>
    </attack>
  </participants>
</evAlert>
```

The ***evAlert*** element contains:

- ***evAlert*** attributes:
  - ✓ ***eventId***: unique identifier
  - ✓ ***severity***: risk level (info/low/med/high
- ***originator*** element: application instance that generates the alert, has 3 child elements, i.e.:
  - ✓ ***hostId***: unique identifier for the host
  - ✓ ***appName***: name of the application
  - ✓ ***appInstanceId***: process identifier
- ***time*** element: event timestamp, i.e. the number of non-leap seconds that have elapsed since 00:00:00 January 1, 1970 UTC

  - ✓ ***offset***: time offset from UTC
  - ✓ ***timeZone***: time zone
- ***interfaceGroup***: network interface group of the alert's traffic
- ***vlan***: vlan number for the alert's traffic
- ***signature***
  - ✓ ***sigId***: signature identifier
  - ✓ ***subSigId***: sub classification identifier of the signature
  - ✓ ***sigName***: signature name
  - ✓ ***version***: signature version
- ***participants*** element: consists of ***attack*** element that contains both the ***attacker*** and ***victim*** information, i.e. the IP address (***addr*** element) and its position (***locality*** attribute, whether inside or outside the network)

According to Cisco Secure Encyclopedia, this signature fires when an ICMP datagram that has a size of greater than 1024 bytes is detected on the network.

Although I can't find the same alert on the on-campus IDS (snort 2.2.0), I do find the same alert with reversing source and destination IPs within a very close timestamp. This turns out to be a large echo reply packet. Below is the corresponding Snort alert.

```
[**] [1:499:4] ICMP Large ICMP Packet [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/09/04-08:56:03.891020 0:D0:0:2C:CF:F5 -> 8:0:7:AF:AA:4B
type:0x8100 len:0x5EE
64.91.255.158 -> X.X.X.X ICMP TTL:50 TOS:0x0 ID:32535 IpLen:20
DgmLen:1500
Type:0  Code:0  ID:39612  Seq:57072  ECHO REPLY
[Xref => http://www.whitehats.com/info/IDS246]
```

A Snort alert contains:

```
[**] [1:499:4] ICMP Large ICMP Packet [**]
```

→ **[**] [*Generator ID*: *Signature ID*: *Revision Number*] *Signature Message* [**]**

```
[Classification: Potentially Bad Traffic] [Priority: 2]
```

→ **[Classification: *Classification Type's Short Name*] [Priority: *Priority level*]**

```
08/09/04-08:56:03.891020 0:D0:0:2C:CF:F5 -> 8:0:7:AF:AA:4B   type:0x8100
len:0x5EE
```

→ ***Timestamp Source MAC Address -> Destination MAC Address***
→ **type: *encapsulation protocol (0x8100 = IEEE 802.1 Q VLAN tagging)***
→ **len: *length of the frame (0x5EE = 1280 + 224 + 14 = 1518)***

```
64.91.255.158 -> X.X.X.X ICMP TTL:50 TOS:0x0 ID:32535 IpLen:20 DgmLen:1500
```

→ ***Source IP Address -> Destination IP Address***
→ ***Protocol ID***
→ **TTL: *time to live***
→ **TOS: *type of service***

→ **ID:** ***IP***
→ **IpLen:** ***length of IP header***
→ **DgmLen:** ***total length of datagram***

```
Type:0  Code:0  ID:39612  Seq:57072  ECHO REPLY
```

→ **Type:** ***ICMP type (because this is an ICMP packet)***
→ **Code:** ***ICMP code***
→ **ID:** ***ICMP identifier***
→ **Sequence:** ***ICMP Sequence Number***
→ ***ICMP packet name***

[Xref => http://www.whitehats.com/info/IDS246]

→ **Reference information**

As shown in the above sample alert, all traffic in the network is encapsulated within IEEE 802.1q frame and thus,all packets in this detect contains VLAN tag. When viewing with ***tcpdump***, the IP header is preceded with ***Vlan ID (ox0006 for Vlan #6)*** and frame type (***0x0800*** for IP frame).

According to Snort documentation[22], a ***Large ICMP*** signature alerts when a packet with payload size (***dsize***) greater than 800 bytes is received from the external network. Because the packet that triggers alert on the border IDS is sourced from the internal network, it does not fire this snort rule. Instead, an alert is generated on the reply packet of the same ICMP packet. Below is the corresponding snort rule.

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Large ICMP
Packet"; dsize:>800; reference:arachnids,246; classtype:bad-unknown;
sid:499; rev:4;)
```

Probability the source address was spoofed

I believe that the probability the source address, i.e. X.X.X.X was spoofed is zero.

As will be discussed in the next section, these large ICMP (echo request) packets are normal packets that come from a live web server on campus. Furthermore, it has an open proxy that is abused by many attackers around the world. Since these large ICMP packets precede each outbound http connections that are initiated by the open proxy users, I am sure that this source address is not spoofed.

Description of attack:

Large ICMP packets are often used in denial of service (DOS) attacks. A popular example is ***Ping o' Death***: Many operating system either crash, freeze, or reboot when receiving oversized IP packets (larger than 65,535 bytes). The existence of these packets is possible because of the IP fragmentation concept. Packets

[22] http://www.snort.org/snort-db/sid.html?sid=499

that are larger than the Maximum Transfer Unit (MTU) size (e.g. 1500 bytes for Ethernet) are transmitted in smaller packets called fragments that are then reassembled by the receiver. All fragments in a fragment train share the same fragment ID, use the fragment offset value to determine its relative position within the train, have a fragment length to indicate the length of data payload it contains, and specify whether there are more fragments following it. An attacker can craft the last fragment of a fragment train such that (***offset + size***) > ***65535***.[23] A variation of Ping o' Death includes ***Jolt.***

Additionally, large ICMP packets also frequently serve as covert channels in DDOS attacks, such as ***Stacheldraht***. This DDOS attack has a 3-tier network architecture that consists of ***attacker***, ***masters/handlers/controllers***, and ***agents***. An ***attacker*** usually has several handlers that control a large set of ***agents***. The large ICMP packets are used for communication between the ***masters/handlers*** and their ***agents*** either to check each other's status or for the ***masters*** to commands the ***agents***.[24]

The CVE numbers related to these large ICMP attacks are ***CVE-1999-0128***[25] (***Ping o' Death***) and ***CAN-1999-0345***[26] (***Jolt ICMP***).

However, large ICMP packets are not always evil. There are cases where they appear as part of normal traffic that comes from systems, such as HP-UX systems with PMTU discovery configured, Windows 2000 systems for determining the speed of the link when utilizing domain controllers, and several load balancing application in determining the most efficient route.[22] Furthermore, Mac OS X also appears to send these large ICMP packets in its normal behavior.[27]

As the payload of the large ICMP packets in this detect consists of ***0x00*** – do not contain any useful data – with the ***Don't Fragment (DF)*** flag set and no fragment offset value (thus, not a fragment), I can pretty sure that they are not ***evil*** packets. This is because most of the large ICMP attacks above utilize either IP fragmentation technique (requires fragments) or non-zero payload (contains commands or certain strings for covert channel communication).

Below is a sample of the large ICMP packets in this detect (viewed with tcpdump).

```
08:56:03.814070 802.1Q vlan#6 P0 150.135.28.50 > 64.91.255.158: icmp:
echo request (DF)
      0x0000   0006 0800 4500 05dc 4401 4000 ff01 XXXX
      0x0010   XXXX XXXX 405b ff9e (0800 7e52 9abc def0)
      0x0020   0000 0000 0000 0000 0000 0000 0000 0000
```

[23] http://www.insecure.org/sploits/ping-o-death.html
[24] https://www.sans.org/resources/malwarefaq/stacheldraht.php
[25] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0128
[26] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0345
[27] http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00062.html

```
0x0030   0000 0000 0000 0000 0000 0000 0000 0000
<snip>
0x05c0   0000 0000 0000 0000 0000 0000 0000 0000
0x05d0   0000 0000 0000 0000 0000 0000
```

7. As mentioned earlier, the IP header is preceded with the VLAN ID, i.e. vlan#6 (***0x0006***) and frame type of IP (***0x0800***)
8. Total packet length (2nd – 3rd offset) = ***0x05dc*** = ***1500*** bytes
9. The 4th and 5th bytes offset of IP header – ***IP ID*** = ***0x4401*** = ***17409***
10. The ***Don't Fragment*** flag – found in the 3 high-order bits of the 6th bytes offset of IP header – is set. The fragment offset value is zero, obtained by combining the rest of the 6th byte offset (exclude the IP flags) and the 7th byte offset.

```
Hex      => Decimal  => IP Flags | Fragment Offset
                        6th offset       | 7th offset
                        xDM      |
0x4000   => 64       => 010      | 00000 00000000
```

IP Flags:
x – reserved, set to 0; D – Don't Fragment; M – More Fragment

11. The 9th byte offset of the IP header – ***Protocol*** = ***0x01*** (i.e. ***ICMP***)

ICMP Header – (0800 7e52 9abc def0)

1. Type = 0x08 = Echo request
2. Code = 0x00
3. Checksum = 9abc def0

If these are not evil packets, what are they? I then decide to passively fingerprint the ***X.X.X.X*** using ***nmap***.

```
# nmap -sS -O X.X.X.X
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on X.edu (X.X.X.X):
(The 65529 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
80/tcp     open        http
407/tcp    open        timbuktu
1080/tcp   open        socks
6667/tcp   filtered    irc
8000/tcp   open        unknown
Remote operating system guess: HP9000 Model 804 K450 running HP/UX
11.00
```

Nmap guesses that ***X.X.X.X*** is an HP-UX machine. If this is true, then those large ICMP packets are most likely Path MTU discovery packets. However, communication with the owner of the machine determines that X.X.X.X is a Mac OS 9 system. Based on existing postings that large ICMP packets are normal for Mac OS X, I assume that they also apply to Mac OS 9.

Furthermore, the fact that these Large ICMP packets make to the top 10 of our daily border IDS report attract my attention to research more on this detect although these large ICMP packets look normal. I capture all inbound and outbound traffic of X.X.X.X for approximately 1 minute and obtain a total of 849 packets. There are 28 large ICMP packets among those 849 packets.

```
# Count the number of icmp packets that have total length > 800 bytes
# tcpdump -nn -r largeicmp 'vlan and icmp and ip[2:2] > 800' |wc -l
  28
```

It seems that X.X.X.X sends a ***ICMP echo request*** packet before initiating an http connection. As previously reported[27], the http connection is started without waiting for the ***echo reply***.

```
08:55:10.739617 802.1Q vlan#6 P0 X.X.X.X > 66.28.56.192: icmp: echo request (DF)
08:55:10.739618 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: S
486205932:486205932(0) win 32768 <mss 1460,wscale 0,nop>
08:55:10.815193 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: S
2966128602:2966128602(0) ack 486205933 win 1460 <mss 1460,nop,wscale 0> (DF)
08:55:10.828310 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: . ack 1 win
32768
08:55:10.834184 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: P 1:544(543)
ack 1 win 32768
08:55:10.910386 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: . ack 544 win
6516 (DF)
08:55:10.914884 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: F 295:295(0)
ack 544 win 6516 (DF)
08:55:10.914887 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: P 1:295(294)
ack 544 win 6516 (DF)
08:55:10.968102 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: . ack 295 win
32768
08:55:11.427569 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: F 295:295(0)
ack 544 win 6516 (DF)
08:55:11.429194 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: . ack 296 win
32768
08:55:11.431946 802.1Q vlan#6 P0 X.X.X.X.60286 > 66.28.56.192.80: F 544:544(0)
ack 296 win 0
08:55:11.506524 802.1Q vlan#6 P0 66.28.56.192.80 > X.X.X.X.60286: . ack 545 win
6516 (DF)
```

As reported by the ***nmap*** result earlier, X.X.X.X is most probably a web server (listening on TCP 80). This is confirmed by being able to browse to ***http://X.X.X.X***, which appears to be an official website of a course and is using WebSTAR/4.1 as the web server.

```
$ telnet X.X.X.X 80
Trying X.X.X.X...
Connected to X.X.X.X.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: WebSTAR/4.1 ID/72833
Connection: Close
Date: Tue, 17 Aug 2004 02:57:17 GMT
```

```
Content-Type: text/html
Content-Length: 3941
Last-Modified: Wed, 14 Apr 2004 15:59:16 GMT
```

Unless this web server also acts as a proxy server, it is seems very rare for a web server to initiate so many outgoing http connections. As ***TCP 8000*** also appears to be listening (in the above ***nmap*** result) and it's known as WebSTAR's default proxy port, I am pretty sure that ***X.X.X.X*** is also a proxy server. The proxy traffic that occurred right before the above ***ICMP echo request*** and http connections is showed as follows.

```
08:55:09.903502 802.1Q vlan#6 P0 80.185.226.226.2168 > X.X.X.X.8000: S
4223262774:4223262774(0) win 16384 <mss 1414,nop,nop,sackOK> (DF)
08:55:09.906879 802.1Q vlan#6 P0 X.X.X.X.8000 > 80.185.226.226.2168: S
485869740:485869740(0) ack 4223262775 win 32768 <mss 1414>
```

The next question is whether this is a legitimate web proxy server or an open web proxy. Based on the variety of addresses (23 addresses from all over the world) connecting to this proxy server from the 1 minutes packet dump and the existence of 1054 unique destination addresses of the large ICMP packets from X.X.X.X on that particular day (based on query result from our border IDS database), I tend to believe it's an open proxy.

```
# List all IP addresses that try to / use X.X.X.X's web proxy
# tcpdump -nn -r largeicmp 'vlan and dst port 8000 and dst host X.X.X.X' | awk
  {'print $5'} | awk -F \. {'print $1 "." $2 "." $3 "." $4'} | sort | uniq -c |
  sort -rn
     18 221.219.71.209 -> China Network Communications Group Corporation
     16 218.11.234.204 -> CNCGROUP Hebei province network -China
     14 218.62.75.62 -> CNCGROUP jilin province network
      6 81.220.247.0 -> Nantes 1 - DHCP - France
      5 83.90.237.150 -> TDC Bredbaand Professional users - Denmark
      5 80.185.226.226
      5 80.116.69.123
      5 68.233.66.182
      5 63.194.24.37
      5 218.22.141.163
      5 203.181.3.250
      4 69.150.134.95
      4 61.149.133.116
      4 211.158.124.12
      3 84.65.34.135
      3 67.68.137.172
      2 68.156.175.13
      2 67.33.171.192
      2 66.139.40.168
      1 68.194.194.40
      1 218.230.41.11
      1 217.224.244.45
      1 172.165.180.58

# Query Border IDS database on count of unique destination addresses
mysql> select count(distinct victim_addr) as ct from 09_Attack, 09_EventAlert,
Sigs, 09_Victim  where 09_Attack.event_id = 09_EventAlert.event_id AND
09_EventAlert.sensor_id = 09_Attack.sensor_id and 09_EventAlert.SIGID =
Sigs.SIGID and 09_Victim.AttackId = 09_Attack.AttackId and 09_EventAlert.SIGID
= 2151  and inet_ntoa(attacker_addr) = X.X.X.X' ;
```

```
+------+
| ct   |
+------+
| 1054 |
+------+
1 row in set (0.11 sec)
```

Thus, although the large ICMPs in this detect are normal packets, it leads to a detection of an open proxy. A vulnerability note that is related to this issue is ***VU#150227***.[28]

Attack mechanism:

As quoted from Joe Sauver;s presentation titled "The Open Proxy Problem"[29], an ***open proxy*** is *a computer that accepts connections from anyone, anywhere, and forwards the traffic from those connections as if it had originated locally from that host*". Open proxy servers are usually available because of misconfiguration, inherent protocol/ application deficiencies, or conscious decision of running open proxy. The reasons that these proxies are widely '*wanted'* by attacker include hiding the real source address of an attack, initiating attack from numerous odd locations, and accessing illegal materials and recreational web sites.

Knowing that X.X.X.X is an open web proxy, the machine owner is contacted to ensure that he is aware of the problem. Below is the response I receive and some sensitive and inappropriate wordings are not included.

```
> >I checked my server where I run the website for XXX course.  Turns
> > out the proxy port was open and someone was running tons of junk
> >through it.....looked like lots of XXX and XXX stuff from the
> >address data in the proxy log.
> >
> >Thanks for alerting me. They basically overloaded my primary
> >web site. The hard drive was spinning like a top!
```

In this particular case, it appears that the proxy server becomes open and prone to abuses due to *misconfiguration/ lack of configuration* by the administrator as installation/ upgrade of the older versions of WebSTAR (3 and 4) include proxy component that is open by default.

To fix the problem, the machine owner immediately closes the proxy service. However, a total rebuilt of the machine using the newer operating system (Mac OS X instead of Mac OS 9) and web server (WebSTAR V instead of WebSTAR 3 or 4) is strongly recommended considering the limited number of information we have regarding what have been done to the open proxy server by anyone that knows about its existence. Any trojan/backdoor might have been installed on it.

[28] http://www.kb.cert.org/vuls/id/150227
[29] http://www.uoregon.edu/~joe/proxies/open-proxy-problem.pdf

It seems however, that the machine has been used to download materials from some recreational web sites.

Correlations:

Mihai (Mike) Cojocea describes detect on ***Large ICMP*** alerts on Mac OS and concludes that they are normal behavior of Mac OS based on the posting below:
http://archives.neohapsis.com/archives/snort/2002-11/0161.html

The actual posting of his detect is available from:
http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00062.html

Although this detect does not involve Large ICMP attacks, below are links to two popular attacks described briefly in this detect:
Ping Of Death, http://www.insecure.org/sploits/ping-o-death.html
Stacheldraht, https://www.sans.org/resources/malwarefaq/stacheldraht.php

A quite detailed discussion on general open proxy problem is written by Joe Souver and available from:
http://www.uoregon.edu/~joe/proxies/open-proxy-problem.pdf

A good source specific to open proxy problem in Mac OS is reported by Chuck Goolsbee in his article titled “Classic Mac OS Servers Exploited by Spammers ” available from:
http://www.mail-archive.com/tidbits-talk@tidbits.com/msg00071.html

Evidence of active targeting:

In case of the large ICMP packets, there is no evidence of active targeting considering a total of 1054 different destination addresses as showed earlier in this detect.

As X.X.X.X is an open proxy resulted from system misconfiguration and there are connections from 23 different IP address around the world to it within 1 minutes as discussed earlier, I believe that there is an evidence of active targeting on X.X.X.X’s open proxy service.

Severity:
The focus of this ***severity*** section is on X.X.X.X’s open proxy service.

severity = (criticality + lethality)-(system countermeasures + network countermeasures)
= (4+5) – (1+3) = 9 – 4 = 5

Criticality: 4

In this detect, X.X.X.X is an official web server of a course. With the limited information on data that are actually stored on the system, I assign a criticality level of 4 to this system.

Lethality: 5

The system has been successfully used as an open proxy for accessing recreational web sites and probably downloading illegal materials. As there are other damages that can be done through open proxy service such as utilizing to initiate attacks to other machines, the lethality of the open proxy service existence on this system is assigned to 5.

System countermeasures: 1

Without full information on the system security of this machine, I assume that it is quite low especially when the web server is left on its default configuration. In addition, the system is using an older OS version (Mac OS 9) and web server (WebSTAR 4). I therefore assign a value of 1 for system countermeasures.

Network countermeasures: 3

In this ***.edu*** network, the existing network countermeasures include border firewall, border and on-campus intrusion detection system, and a relatively loose ingress and egress filtering at the border (permit all and deny specific ports such as the NetBIOS and worm-related ports) and distribution routers as mentioned in the first section of this detect. Thus, I assign a value of 3 for network countermeasures.

Defensive recommendation:

There are several steps that can be taken to detect open proxy from the network level:

a. Apply more rigid ingress filtering by gradually moving from '***permit all, deny specifics***' to '***permit specifics, deny all***'.
b. Implement distribution level firewalls or reflexive access lists to prevent uninvited inbound traffic from outside the local area network. This way, the exposure of a certain department's network is greatly reduced.
c. Regular audit for possibility of open proxy services within the local area network by the departmental network managers as they know better regarding their local network. This can be performed by simple NMAP scan for popular proxy ports such as TCP 3128, 8080, 6588, 80, 81, 4480 for web open proxy. More specific tools are also available, for example: proxy hunter and proxy sniper.[29]
d. Close monitoring of network traffic utilization and abnormal network activities may help determine the existence of open proxy, as described in this particular detect.

As for the system level prevention and detection:

e. Use the latest operating system and software version and apply all patches / updates especially those are have security implications. As usual, test them before using them in the production environment and always perform backup before performing these upgrades/updates when they relate to critical systems/applications.
f. When proxy server is required, implement access controls to restrict the proxy service to those that need it.
g. Review the default web server and proxy configuration to ensure only necessary services are enabled with controlled exposures.
h. Monitor the web and proxy log closely for abnormal or suspicious activities.

Multiple choice test question:

Which of the following OS is known as causing false positive on Large ICMP alerts?
A) HP-UX
B) Windows 2000
C) Mac OS
D) All true
Answer: D

Detect 3: My Doom M/O

Source of Trace:

> This trace is obtained from the on-campus IDS of an ***.edu*** network. The network diagram for this network is included in ***Detect 2***.

As in most ***.edu*** network, the ingress and egress filtering at the border router and firewall are fairly loose. In this case, only certain popular backdoor and worm activity ports are blocked, including the NetBIOS ports. In addition, the border IDS is configured to issue temporal blocks requests to the border firewall when certain signature alerts are triggered.

Detect was generated by:

This detect is generated by the on-campus IDS (a commercial version of Snort) as it showed up as one of the top 5 signatures in the hourly report. This is particularly interesting because we have not had many of these infections in the last month, or least the one that attracted attention. While Figure 4 provides a summary of source and destination addresses contributing to the ***MyDoom M/O*** alerts within the last hour, Figure 5 shows a sample event view.

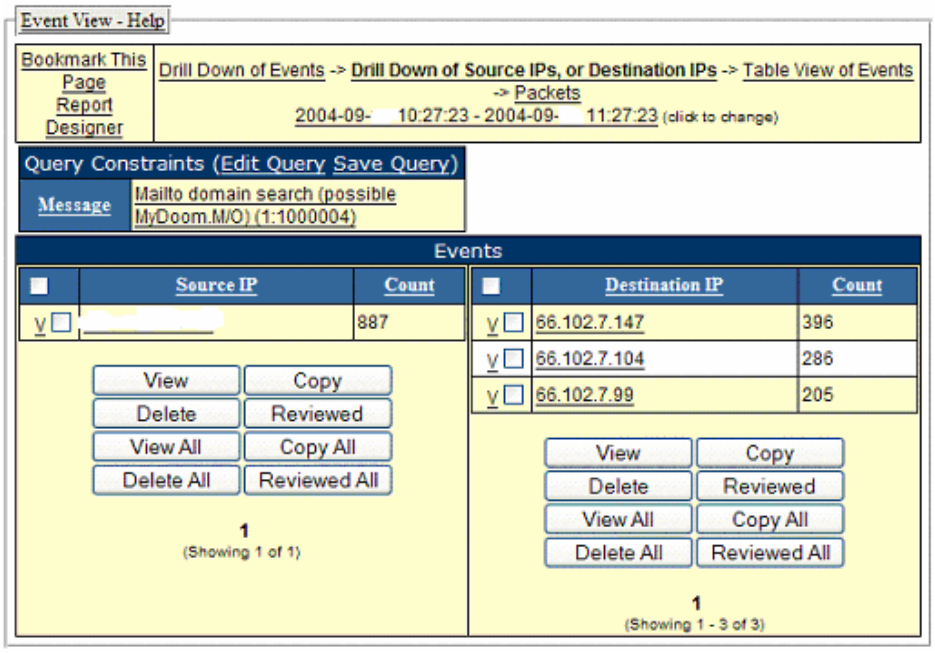


Figure 4 On-campus IDS: Alerts Summary - MyDoom M/O events

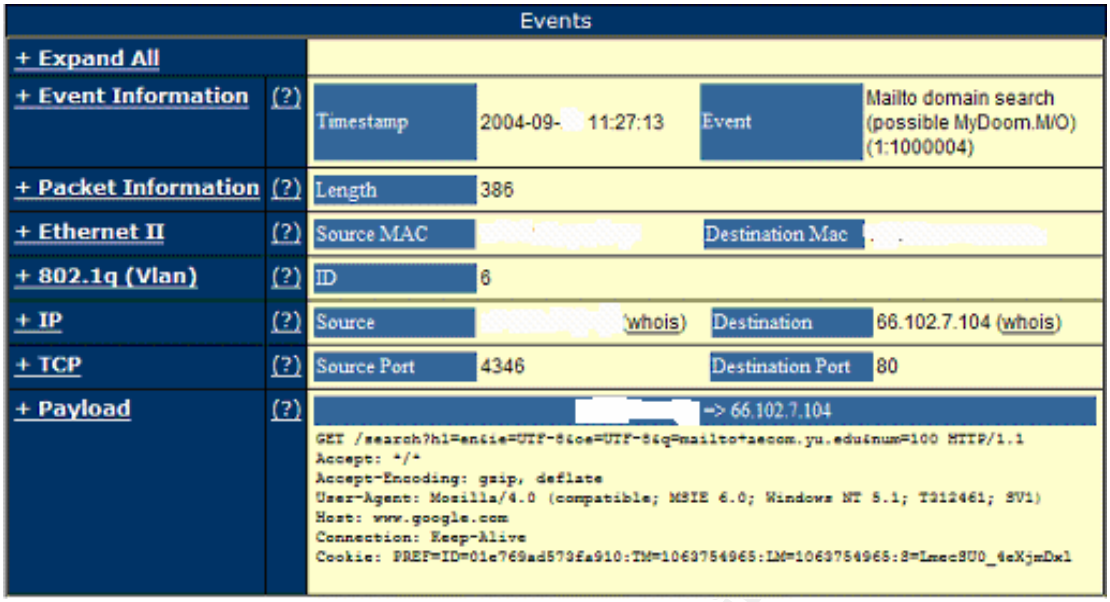


Figure 5 On-campus IDS: Event View - MyDoom M/O event

Since there are 887 alerts generated within the last hour, it means that this particular host performs approximately 15 "***mailto domain search on google.com***" attempts per minute, which does not seem to be a normal behavior.

The query result from the border IDS database – for all alarms sourced from this particular on-campus address in the last hour – appears to confirm that the anomalous activities from ***X.X.X.X***.

Attacker Address	SIGID	Signature Name	Count
X.X.X.X	3135	MyDoom Virus Activity	55
X.X.X.X	3110	SMTP Suspicious Attachment	14

Table 1 Border IDS – Alerts Summary – Attacker: XXX.XXX.31.249

This signature in this detect is obtained from ***Lurhq's Analysis*** on ***Zindos*** worm, a backdoor left by ***MyDoom.M/O*** infections.[30]

```
alert tcp any any -> any 80 (content:"GET /search?hl=en&ie=UTF-
8&oe=UTF-8&q=mailto+"; nocase:; depth:45; content:"Host|3a|
www.google.com"; nocase:; reference:url,www.lurhq.com/zindos.html;
msg:"Mailto domain search \(possible MyDoom.M/O\)"; classtype:trojan-
activity; sid:1000004; rev:1;)
```

This signature alerts when a packet contains:

"***GET /search?hl=en&ie=UTF-8&oe=UTF-8&q=mailto+***" within the 45 bytes of the payload and followed by ***"Host|3a| www.google.com"***

[30] www.lurhq.com/zindos.html

In addition, the content checking is case-insensitive.

Probability the source address was spoofed

I believe that the probability the source address, i.e. X.X.X.X was spoofed is zero.

As will be discussed in the next section, this is an email worm that collects domain names from user files and uses them to querying various search engines for valid email addresses within these domains. Therefore, this type of attack requires established connections with the search engines to gather the email addresses information. Therefore, I am convinced that the source address is not spoofed.

In addition the destination addresses belong to Google Inc. as showed in the Geektools output below:

```
NetRange: 66.102.0.0 - 66.102.15.255
CIDR: 66.102.0.0/20
OrgName: Google Inc.
OrgID: GOGL
Address: 2400 E. Bayshore Parkway
City: Mountain View
StateProv: CA
PostalCode: 94043
Country: US
```

Description of attack and Attack Mechanism:

MyDoom.M/O is a variant of the mass-mailinglist MyDoom that introduces a new means of collecting email addresses via various well-known search engines: Google, Lycos, AltaVista and Yahoo. Once infected, MyDoom.M/O opens a backdoor known as ***Zindos*** that listens on ***TCP*** port ***1034*** for remote connections. It then gathers email addresses from user files that have certain extensions and also use the collected domain names to harvest additional email addresses from ***search.lycos.com, search.yahoo.com, www.altavista.com, www.google.com***.

Even though this signature is only triggered by those domain searches targeted to ***google.com***, I notice that X.X.X.X also tries to use search.yahoo.com in some of the packets I captured around that timeframe. However, none of these attempts to yahoo seems to be successful as they are all redirected to the ***Yahoo! Gone*** page as showed in the 2nd packet dump below.

```
18:10:10.238831 802.1Q vlan#6 P0 X.X.X.X.2567 > 216.109.117.133.80: P
1:333(332) ack 1 win 17520 (DF)
0x0000   0006 0800 4500 0174 0646 4000 8006 XXXX       ....E..t.F@.....
0x0010   XXXX XXXX d86d 7585 0a07 0050 5471 5df2       .....xx....PTq].
0x0020   49a5 8fcb 5018 4470 779e 0000 4745 5420       I...P.Dpw...GET.
```

```
0x0030   2f73 6561 7263 683f 703d 6d61 696c 746f       /search?p=mailto
0x0040   2b6e 6472 6972 6573 6f75 7263 652e 6f72       +ndriresource.or
0x0050   6726 6569 3d55 5446 2d38 2666 723d 6670       g&ei=UTF-8&fr=fp
<snip>
0x0100   0d0a 486f 7374 3a20 7365 6172 6368 2e79       ..Host:.search.y
0x0110   6168 6f6f 2e63 6f6d 0d0a 436f 6e6e 6563       ahoo.com..Connec

18:10:10.351009 802.1Q vlan#6 P0 216.109.117.133.80 > X.X.X.X.2567: P
1:1152(1151) ack 333 win 65535 (DF)
0x0000   0006 0800 4500 04a7 10d7 4000 3306 XXXX       ....E.....@.3...
0x0010   XXXX XXXX 9687 1ff9 0050 0a07 49a5 8fcb       .xx......P..I...
0x0020   5471 5f3e 5018 ffff 512f 0000 4854 5450       Tq_>P...Q/..HTTP
0x0030   2f31 2e31 2034 3130 2047 6f6e 650d 0a44       /1.1.410.Gone..D
<snip>
0x00c0   0d0a 3364 3920 2020 200d 0a3c 6874 6d6c       ..3d9......<html
0x00d0   3e3c 6865 6164 3e3c 7469 746c 653e 5961       ><head><title>Ya
0x00e0   686f 6f21 202d 0a34 3130 2047 6f6e 653c       hoo!.-.410.Gone<
0x00f0   2f74 6974 6c65 3e3c 2f68 6561 643e 3c62       /title></head><b
```

While I do not have information on whether any attempts to other search engines are successful, I can be pretty sure based on the alerts fired on our border IDS regarding ***MyDoom Activity*** and ***SMTP suspicious attachment*** that is a real infected machine. In addition, the system appears to be down when I try to scan for open port ***TCP 1034***.

Correlations:

A detail analysis of MyDoom M/O and its backdoor can be found at:
http://www.lurhq.com/zindos.html
http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.m@mm.html

Evidence of active targeting:

There is an evidence of active targeting since the "***mailto domain search***" requests are only sent to: ***search.lycos.com, search.yahoo.com, www.altavista.com, www.google.com***. There are also several online postings regarding the effect of this particular worm on the performance of these search engines when it first showed up in wild last July.[31]

Severity:
severity = (criticality + lethality)-(system countermeasures + network countermeasures)
= (1+3) – (1+3) = 0

Criticality: 1

[31] http://news.com.com/Google%2C+other+engines+hit+by+worm+variant/2100-1023_3-5283750.html

In this detect, X.X.X.X is an end-user workstation, therefore the criticality of the system only concerns a single user. I assign a criticality level of 1 to this system.

Lethality: 3

The system has been successfully used as an infected by ***MyDoom M/O***. Although infection of other hosts still requires user contact on opening the malicious email attachment, I assign the lethality level to 3 considering the relatively low user awareness on risks associated with opening unknown emails.

System countermeasures: 1

Without full information on the system security of this machine, I assume that it is quite low especially because it does not seem to run the latest anti-virus update that should be able to detect and disinfect MyDoom M/O. I therefore assign a value of 1 for system countermeasures.

Network countermeasures: 3

In this ***.edu*** network, the existing network countermeasures include border firewall, border and on-campus intrusion detection system, and a relatively loose ingress and egress filtering at the border (permit all and deny specific ports such as the NetBIOS and worm-related ports) and distribution routers as mentioned in the first section of this detect. Thus, I assign a value of 3 for network countermeasures.

Defensive recommendation:

Network level defenses:
- ✓ Regularly review the IDS logs to identify infected systems

As for the system level prevention and detection:
- ✓ Have the anti-virus software installed with remote/live update capabilities enabled
- ✓ User education on the dangerous of opening unknown email attachments

Multiple choice test question:

Which of the following is not the target search engine for MyDoom.M/O?
A) Google
B) Altavista
C) AskJeeves
D) Yahoo

Answer: C

Part 3 – Analyze This

Executive Summary

As one of the efforts to achieve a more secure campus network, MY.EDU has put in place Snort Intrusion Detection Systems (IDS) on its network perimeter to monitor both the incoming and outgoing traffic for suspicious, malicious, and bad traffic. Due to the nature of university networks that tend to be relatively open and highly distributed, it is very common to encounter a lot of false alarms from the IDS sensor. During this analysis, several configuration changes are noted and recommended. This will hopefully reduce the number of false alarms that need to be reviewed by the university's IDS analysts. In addition, a list of hosts that are possibly infected by the Phatbot/Agobot worm is also provided for further investigation by the system owner or administrator.

While 'closing the border' might sound like a 'too good to be true' idea, the university should consider putting together an action plan toward this objective in the long term. This should include the creation of security policy and procedure and end-user awareness programs in addition to building layers of security defenses. Furthermore, the University should also reconsider its existing policy regarding P2P traffic due to the increased amount of bandwidth it consumes and its security and legal risks.

Despite the high number of false alarms and a list of possible worms infected hosts, the University does appears to maintain adequate protection on its critical resources. One example is the use of current version of ***Sendmail*** on the campus main mail servers.

Although detail recommendations are provided at the end of each analysis section, below is a quick list of the recommendations:

Snort Configurations

1. Several Snort custom signatures, especially ***MY.NET.30.3 activity*** and ***MY.NET.30.4 activity***, are too generic in nature and require modifications to reduce the number of false alarms.
2. It appears that an old Snort fragmentation preprocessor is still used and triggers false alarms. The old ***defrag*** preprocessor should be replaced with ***frag2***.
3. Exclude the campus DNS servers from triggering port-scan alerts.

Perimeter Protections

Consider modifying the egress and ingress filtering at the perimeter router/firewall to:

1. Block any incoming and outgoing traffic destined to Windows NetBIOS ports: ***UDP*** and ***TCP 135-139,445***
2. Drop any incoming and outgoing traffic related to the ***'local link'*** IP ranges (***169.254.0.0/16***)

Others

1. Regular audit of the campus network for vulnerable systems.
2. Implement a network access control mechanism to ensure that only patched and clean systems can be connected to the network. This especially applies to Microsoft Windows systems.
3. Consider using packet shaper devices to manage the amount of P2P traffic.
4. Improve end-user awareness on computer security matters through campus security awareness campaign.

Suspicious internal machines

Below is a list of internal machines that require immediate attentions. If possible, disconnect them from the network and contact the system owners for worm disinfections.

Possible Agobot Infection

MY.NET.153.174	MY.NET.97.57	MY.NET.97.78	MY.NET.98.53
MY.NET.153.195	MY.NET.97.124	MY.NET.97.49	MY.NET.81.59
MY.NET.111.51	MY.NET.97.103	MY.NET.97.235	MY.NET.84.235
MY.NET.97.169	MY.NET.97.43	MY.NET.97.92	MY.NET.153.90
MY.NET.97.25	MY.NET.97.30	MY.NET.190.92	MY.NET.70.96
MY.NET.98.65	MY.NET.97.159	MY.NET.97.129	MY.NET.71.235
MY.NET.97.12	MY.NET.97.126	MY.NET.97.108	MY.NET.153.99

Possible Opaserv Infection

MY.NET.150.44	MY.NET.150.198

Table 2 below lists all logs analyzed in this report.

Alert logs	**Scan Logs**	**OOS Logs**
alert.040327	scans.040327	oos_report_040327
alert.040328	scans.040328	oos_report_040328
alert.040329	scans.040329	oos_report_040329
alert.040330	scans.040330	oos_report_040330
alert.040331	scans.040331	oos_report_040331

	Count		
	Alert	**Scan**	**OOS**
Raw records	922977	22581005	3996
Analyzed	53482 (mysql db)		

records	777150 (txt file)	22580794	3994

Table 2. Log Files Summary

Count of Alert by Signature

Alert name	Alert Counts								
	Total	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I to I	E to E
MY.NET.30.3 activity	28203	198	1			28203			
MY.NET.30.4 activity	21295	284	1			21295			
High port 65535 tcp - possible Red Worm - traffic	13421	69	34	34	103	7025	6396		
EXPLOIT x86 NOOP	9343	796	618			9343			
Incomplete Packet Fragments Discarded	5357	83	88			5357			
SMB Name Wildcard	5164			156	613		5164		
Null scan!	1304	199	109			1304			
High port 65535 udp - possible Red Worm - traffic	1239	44	21	8	19	676	563		
TFTP - Internal UDP connection to external tftp server	1157	5	6	1	3	1153	4		
Traffic from port 53 to port 123	1154	2	2			1154			
NMAP TCP ping!	931	173	81			931			
[UMBC NIDS IRC Alert] IRC user /kill detected - possible trojan.	616	51	62			616			
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	479			27	2		479		
Possible trojan server activity	352	31	17	14	34	159	193		
SUNRPC highport access!	339	25	21			339			
TFTP - Internal TCP connection to external tftp server	232	2	2	1	1	174	58		
Tiny Fragments - Possible Hostile Activity	205	12	11	1	1	159	46		
TFTP - External TCP connection to internal tftp server	171	4	49	30	4	95	76		
TCP SRC and DST outside network	157	39			80				157
FTP passwd attempt	133	106	1			133			
[UMBC NIDS] External MiMail alert	133	17	1			133			
TCP SMTP Source Port traffic	128	4	1			128			
External RPC call	108	2	97			108			
SMB C access	93	18	5			93			
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	89			2	52		89		
ICMP SRC and DST outside network	67	22			67				67
IRC evil - running XDCC	57			6	6		57		
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	52			2	2		52		
NIMDA - Attempt to execute cmd from campus host	50			5	39		50		
FTP DoS ftpd globbing	41	8	1			41			
EXPLOIT x86 setuid 0	37	29	23			37			
Attempted Sun RPC high port access	35	11	15			35			
RFB - Possible WinVNC - 010708-1	35	3	6	12	5	15	20		
DDOS shaft client to handler	25	4	2			25			
EXPLOIT x86 setgid 0	24	18	17			24			
EXPLOIT NTPDX buffer overflow	20	12	9			20			
EXPLOIT x86 stealth noop	13	10	7			13			
connect to 515 from outside	11	1	2			11			
SYN-FIN scan!	11	4	5			11			
[UMBC NIDS IRC Alert] Possible	11	1	3			11			

drone command detected.								
Probable NMAP fingerprint attempt	10	7	7			10		
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	10	4	4			10		
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	9	4	6			9		
DDOS mstream client to handler	5	3	3			5		
External FTP to HelpDesk MY.NET.53.29	5	3	1			5		
External FTP to HelpDesk MY.NET.70.50	3	3	1			3		
EXPLOIT x86 NOPS	2	2	2			2		
External FTP to HelpDesk MY.NET.70.49	2	2	1			2		
NIMDA - Attempt to execute root from campus host	2			2	2		2	
[UMBC NIDS] Internal MiMail alert	2			2	2		2	
NETBIOS NT NULL session	1	1	1			1		
[UMBC NIDS IRC Alert] K:lined user detected\	1	1			1			1
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	1	1	1			1		
Totals:	**92345**	**2318**	**1345**	**303**	**1036**	**78869**	**13251**	**225**

Table 3. Alert Logs: Alerts Summary

TOP 10 Attack Participants

External Attacker		Internal Attackers		Internal Victims	
IP Address	Count	IP Address	Count	IP Address	Count
68.55.174.94	7590	MY.NET.97.82	5022	MY.NET.30.3	28204
67.31.152.200	6585	MY.NET.11.7	1659	MY.NET.30.4	21295
80.181.112.186	5459	MY.NET.53.111	847	MY.NET.97.82	5460
68.55.178.168	3127	MY.NET.150.44	621	MY.NET.153.176	5179
140.142.8.73	3074	MY.NET.150.198	515	MY.NET.1.3	2858
69.136.228.63	2988	MY.NET.110.72	488	MY.NET.53.111	1239
68.57.90.146	2593	MY.NET.75.13	419	MY.NET.17.4	826
65.107.99.68	2301	MY.NET.190.92	344	MY.NET.12.6	526
69.240.222.54	2201	MY.NET.5.34	159	MY.NET.110.72	451
138.88.183.54	2151	MY.NET.29.30	150	MY.NET.17.3	343

Table 4. Alert Logs: Top 10 Attack Participants

Top Signatures (Alert Counts > 5000)

1. MY.NET.30.3 activity

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
28203	198	1			28203			

Unique Source Ports		Unique Destination Ports	
1719		1344	
Top Ports	Count	Top ports	Count
1078	5486	524	17513

1033	2109	3019	6730
1077	1244	80	288
1034	848	6129	20
60017	809	4899	15
60019	778	21	11
1035	773	4000	11

Table 5. Alert Logs: Top Ports Summary – MY.NET.30.3 activity

This signature seems to alert on any traffic destined to ***MY.NET.30.3***. Most of the traffic is destined to port ***524*** and ***3019*** which are reserved and non-adjustable ports used by the ***NetWare Core Protocol (NCP)*** and ***Novell Distributed Print Services Resource Management Server (NDPS RMS)*** applications respectively on a Novell NetWare 6 server.[32] Both Michael Meacle[33] and Peter Storm[34] also concluded in their GCIA practicals that ***MY.NET.30.3*** is a NetWare system. As ***MY.NET.30.3*** itself seems to be alive, accessing it from the Internet returns the default Netware 6's Enterprise Server installation page. The ***Novell iPrint*** (an extension of NDPS that allows Internet printing) service appears to be installed and available via ***Novell iManager*** authentication.

Due to the nature of this signature, it will generate a lot of alerts for events that might not be harmful. Among the list of top attackers, mostly Comcast users, for this particular signature listed in Table 6 , ***67.31.152.200*** also shows up in the ***Scans*** log as the source address of ***97%*** SYN scans (Table 7) that are directed to ***MY.NET.30.3*** within those 5 days and specifically on ***March 27, 2004*** between ***11:58:21*** and ***12:00:52***. These scans are destined to ***1327*** unique TCP ports that range from ***2/TCP*** to ***65000/TCP***.

Top Attackers		Count
Source IP	**Hostname**	
68.55.174.94	pcp05133469pcs.elkrdg01.md.comcast.net	7559
67.31.152.200	**dialup-67.31.152.200.Dial1.Denver1.Level3.net**	**3560**
68.55.178.168	pcp233959pcs.elictc01.md.comcast.net	2998
68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	2446
69.240.222.54	pcp0010273370pcs.bbridg01.fl.comcast.net	1997
131.92.177.18	aeclt-cf00a4.apgea.army.mil	1966
64.134.68.238	dhcp64-134-68-238.wmc.chi.wayport.net	1383
216.56.88.95	wisc-ip95.mpw.net	1121

Table 6. Alert Logs: Top Attackers – MY.NET.30.3 activity

Attacker IP	Count
67.31.152.200	2092
Other 53 Attacker IPs	65
Total	**2157**
Scan types	**SYN**

[32] http://support.novell.com/cgi-bin/search/searchtid.cgi?/10065719.htm
[33] http://www.giac.org/practical/GCIA/Michael_Meacle_GCIA.pdf
[34] http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf

Table 7. Scan Logs: Port & Scan Type Summary – Victim: MY.NET.30.3

Although there are also four instances of completed port scan alerts (***spp_portscan***) from ***67.31.152.200*** as part of the ***Alerts*** log, their timestamps do not correspond to the ones mentioned earlier. Since ***67.31.152.200*** is also the 2nd top off-campus attacker, there are definitely other alerts generated by this particular host as discussed later in this paper.

Conclusions:
MY.NET.30.3 activity is a very generic signature and tends to generate a lot of false positives. This will require an IDS analyst to spend additional time identifying the false positives.

Recommendations:
1. Tune the signature:
    a. Review the objective of this signature to determine more specific rule parameters.
    b. Modify the ***event thresholding*** and ***event suppression*** parameters to reduce the number of alerts generated.
2. The fact that a signature is created to observe all traffic to ***MY.NET.30.3*** indicates that it is a critical system that requires close monitoring. Therefore, a periodic system-level security audit on the system is recommended to ensure adequate protection (e.g. up-to-date security patches, proper system logging and monitoring, identification and removal of unnecessary services, password policy, etc.) is in place.
3. In addition to HTTPS (HTTP with Secure Socket Layer) protocol in the ***Novell iManager*** authentication page, a valid username/password combination is required. Therefore, a strong password policy is critical to survive brute force attempts.
4. Unless it needs to be accessible by everyone on the Internet, use router access-list or firewall rules to restrict access to those that need it.

2. MY.NET.30.4 activity

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
21295	284	1			21295			

Unique Source Ports		Unique Destination Ports	
3718		1323	
Top Ports	Count	Top ports	Count
42100	403	51443	13582
1339	365	524	3619
1318	352	80	1039
3221	332	6129	23
3201	314	4899	17

1062	257	21	12
2834	196	4000	12
3223	122	20168	10
3979	111	17300	7
3416	71	1080	7

Table 8. Alert Logs: Top Ports Summary – MY.NET.30.4 activity

Similar to the ***MY.NET.30.3 activity*** signature discussed earlier, only this signature alerts on all traffic destined to ***MY.NET.30.4***. The three top destination ports for this signature are 51443, 524, and 80, which are consistent with the previous reports by Michael Meacle[33] and Pete Storm[34].
Again, visiting the actual MY.NET.30.4 system via browser reveals that it is the campus' ***Novell NetStorage*** introduction page. It appears that ***NetStorage*** is used to provide web access to the NetWare network shares through port 51443, which is the default Apache Web Server's HTTPS port when NetWare Enterprise Server is installed and ***NetStorage*** requires this same Apache port[35].

Attacker IP		Count
IP Address	**Hostname**	
69.136.228.63	pcp08652049pcs.towson01.md.comcast.net	2988
67.31.152.200	dialup-67.31.152.200.Dial1.Denver1.Level3.net	2967
138.88.183.54	pool-138-88-183-54.res.east.verizon.net	2151
68.55.191.197	pcp05510211pcs.owngsm01.md.comcast.net	1876
68.55.86.79	pcp04598795pcs.elictc01.md.comcast.net	1459
68.50.102.64	bgp01546912bgs.longhl01.md.comcast.net	1458
134.192.65.152	hshsl152.umaryland.edu	1333

Table 9 . Alert Logs: Top Attackers – MY.NET.30.4 activity

Most of the top attacker addresses above are also local ISP (e.g. Comcast) users, which are consistent with most attacker addresses for the ***MY.NET.30.3 activity*** alerts and with Peter Storm's finding for the same signature in his practical[34]. From these addresses, only ***67.31.152.200*** – the 2nd top off-campus attacker – can be found in the ***Scans*** log. This is the same attacker that triggers a lot of ***MY.NET.30.4 activity*** alerts discussed earlier and appears to be responsible for 96.6% of the **SYN** scanning traffic destined to ***MY.NET.30.4*** as showed in Table 10. All these scanning activities occur on March 27, 2004 between ***12:00:51*** and ***12:03:22*****,** this means that these activities follow the scans directed to ***MY.NET.30.4*** almost immediately.

Attacker IP	**Count**
67.31.152.200	1892
Other 53 Attacker IPs	67
Total	1959
Scan types	**SYN**

Table 10 . Scan Logs: Port and Scan Type Summary – Victim: MY.NET.30.4

[35] http://www.leu.bw.schule.de/netze/novell/ml2/patches/nw6sp3.txt

Conclusions:
MY.NET.30.4 activity is another very generic signature that requires tuning.

Recommendations:
In addition to recommendations stated earlier for the ***MY.NET.30.3 activity*** signature alerts, both the ***MY.NET.30.3 activity*** and the ***MY.NET.30.4 activity*** signatures might be combined to create a more specific signature.

3. High port 65535 tcp - possible Red Worm - traffic

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
13421	69	34	34	103	7025	6396		

Unique Source Ports		Unique Destination Ports	
17		15	
Top Ports	Count	Top ports	Count
65535	7195	65535	6226
1122	5022	1122	5459
3658	847	3658	1228
25	110	25	268
80	84	80	92
110	66	110	50
2757	35	113	36
443	22	443	23
113	16	2757	14
5677	9	4662	10
4662	8	5677	10
143	2	143	2
38057	1	38057	1
10182	1	6346	1
10183	1	94	1
6346	1		
94	1		

Table 11 . Alert Logs: Ports Summary – High port 65535 tcp - possible Red Worm - traffic

This is another custom signature that is configured to trigger on incoming traffic that destines to or is sourced from TCP port ***65535***. This is confirmed by adding the number of alerts that have TCP 65535 as either the source or destination port that returns this signature's total alerts: ***13421***. As suggested by its alert message, this signature is intended to detect ***Red Worm/ Adore*** Worm infection. Infected machines open a backdoor on TCP 65535 after receiving a ping packet with a correct size[36] and scan randomly generated Class B subnets for vulnerable systems on port ***53/bind***, ***111/statdx***, and ***515/lpmg***. I therefore compare the IP addresses of the ***Alerts*** and ***Scans*** logs to obtain ***Red Worm***

[36] http://www.f-secure.com/v-descs/adore.shtml

infected hosts that might also be scanning on either of these ports. ***210.139.118.246*** appears to be the only matching address and it exists in the ***Scans*** log as sending one UDP packet with destination port 53 to MY.NET.1.3 (confirmed as MY.NET's name server based on ***Geektools*** search output). Thus, I can not any active scanning evidence on any of those 3 ports coming from addresses related to these signature alerts.

Furthermore, this worm also tries to mail the infected system's IP address to ***adore9000@21cn.com*** & ***adore9000@sina.com*** or ***adore9001@21cn.com*** & ***adore9001@sina.com***.[37]

Because ***TCP 65535*** is within the legitimate ephemeral ports range (1024-65535) that can be randomly picked by client programs to connect to the servers, this signature can trigger false alarms quite easily, especially those involving ***well-known*/ *server*** ports (***1-1023***) such as port ***25*** (***smtp***), ***80*** (***http***), ***110*** (***pop3***), ***143***(***imap***), and ***443***(***https***)[38] listed in Table 11. In addition, alerts related the p2p ports (***4662/edonkey*** & ***6346/gnutella***) are most likely also false positives.

Source IP	Source Port	Destination IP	Destination Port	Count
80.181.112.186	65535	MY.NET.97.82	1122	5459
MY.NET.97.82	1122	80.181.112.186	65535	5022
66.118.165.120	65535	MY.NET.53.111	3658	1228
MY.NET.53.111	3658	66.118.165.120	65535	847
MY.NET.60.17	110	68.55.62.110	65535	66
<snip>				
202.33.252.164	65535	MY.NET.97.82	113	1

Table 12 . Alert Logs: Top Participants Summary – High port 65535 tcp - possible Red Worm - traffic

As TCP 113 (***identd trojan/auth***) has been used a lot recently by various worms including variants of Korgo[39] and Rbot[40], any on-campus Windows machines that listen on this port should be investigated, i.e. MY.NET.97.82 in this case because it appears in one of this signature's alerts as a target host with target port TCP 113. Although there are 94 alerts in the ***Scans*** log sourced from this IP, none of the destination ports seems to be related to the latest ****Bot*** worm activities. Therefore, this is most probably another false alarm.

Although most of the other ports are registered to various services and there are no known vulnerabilities related to these ports from dshield.org port reports, it's hard to determine whether they do provide legitimate services without better host-level information. This is particularly because they are not as common services as the ones listed above

[37] http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf
[38] http://www.iana.org/assignments/port-numbers
[39] http://securityresponse.symantec.com/avcenter/venc/data/w32.korgo.f.html
[40] http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=39437

- TCP ***1122*** (***availant-mgr*** Availant Manager)
- TCP ***3658*** (***ps-ams*** PlayStation AMS (Secure))
  Should this is a legitimate service, the destination address of 66.118.165.120 is actually registered to Sago Networks that provides "*fully managed preconfigured game servers*".[41]
- TCP ***2757*** (***cnrp*** Common Name Resolution Protocol)
- TCP ***94*** (***objcall Tivoli Object Dispatcher***)
- TCP ***5677,38057, 10182, 10183***

I notice two source addresses (***MY.NET.12.6 and MY.NET.12.4***) of this signature also appear as source addresses in the OOS Logs. Although the timestamps do not seem to match, they share the same source ports, i.e. ***TCP 25*** and ***143*** respectively. Connecting to these machines on the corresponding port confirms that that they have either ***Sendmail*** (***MY.NET.12.6***) or ***imap*** services (***MY.NET.12.4***) installed.

Conclusion:
This signature does not function effectively in detecting the ***Red Worm*** activities.

Recommendations:
- Further research on alerts that are related to other epheremal ports, especially on the 2 top ports: TCP ***1122*** and ***3658***.
- Always keep up with the latest security updates/patches.
- Scan campus to identify vulnerable version of BIND, rpc.statd, LPRng, and wu-ftp
- Encourage owners of mail servers, block and log attempts from machines that send emails to adore9000@21cn.com, adore9000@sina.com, adore9001@21cn.com, and adore 9001@sina.com.[37]
- Log and block access to the go.163.com domain[37]
- Consult http://www.sans.org/y2k/adore.htm on Adore/Red Worm detection and removal *how-to*
- Use more specific signature (http://whitehats.com/info/IDS457)

```
alert TCP $EXTERNAL any -> $INTERNAL 515 (msg: "IDS457/lpr_LPRng-
redhat7-overflow-security.is"; flags: A+; content: "|31DB 31C9 31C0
B046 CD80 89E5 31D2 B266 89D0 31C9 89CB|"; nocase; classtype:
system-attempt; reference: arachnids,457;)
```

4. EXPLOIT x86 NOOP

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
9343	796	618			9343			

Unique Source Ports	Unique Destination Ports

[41] **http://www.sagonet.com/servers/gaming.php**

1349		69	
Top Ports	Count	Top ports	Count
2247	133	80	8216
1777	127	1025	550
4674	125	135	278
4954	124	119	108
4669	122	8881	60
4671	122	445	14
4522	120	6129	13
4672	119	2032	12
3767	117	3338	10
4756	116	3315	8

Table 13. Alert Logs: Ports Summary – Exploit x86 NOOP

While there is no exact matching Snort signature for ***EXPLOIT x86 NOOP***, it appears to be closely related to Snort's SID 648[42] (***SHELLCODE x86 NOOP***) that detects a series of Intel's x86 NOP instructions. Buffer overflow attacks utilize these instructions – that perform null instructions – to pad the front of the overflow buffer and thus, increase the chances of successful attacks. An Intel's NOP instruction is translated to 0x90.[43] This signature is known as generating many false alarms because the x86 NOP is often found in large file such as image transfer traffic.

Based on the top destination ports summarized in Table 13, port ***80*** (***http***) is the target port for 88% of the total alerts. Due to the increased number of file transfer through web browser, I tend to believe that these are false positives.

As for the 2nd top targeted port, ***1025***, Table 14 and Table 15 list its registered services and known vulnerabilities. Since this port is related to several known Trojans and a LSASS buffer overflow vulnerability, further analysis on the on-campus victims by correlating to the ***Scans*** log is necessary to determine possible compromised hosts. This LSSAS vulnerability is addressed by Microsoft with Security Bulletin MS04-011 and has been wildly exploited through numerous worms including ***Phatbot/Agobot***, ***Korgo***, and ***Sasser***. The machines infected by variants of these worms will actively scan the network for other vulnerable machines on various TCP ports including 80, 135, 139, 445, 1025, 1434, 2745, 3127, 3410, 5000, 6129.[44] This topic will be discussed more in the ***Scans Log*** section.

Protocol	Service	Name
tcp	blackjack	network blackjack
tcp	FraggleRock	[trojan] Fraggle Rock
tcp	listen	listener RFS remote_file_sharing
tcp	md5Backdoor	[trojan] md5 Backdoor
tcp	NetSpy	[trojan] NetSpy
tcp	RemoteStorm	[trojan] Remote Storm

[42] http://www.snort.org/snort-db/sid.html?sid=648
[43] http://www.insecure.org/stf/smashstack.txt
[44] http://isc.sans.org/diary.php?date=2004-04-30

udp	blackjack	network blackjack
udp	RemoteStorm	[trojan] Remote Storm

Table 14 Dshield Port Report: Port 1025 Services[45]

CVE ID	Pro-tocol	Src Port	Target Port	Description
CAN-2003-0533	tcp	any	1025	Buffer overflow in certain Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows NT 4.0 SP6a, 2000 SP2 through SP4, XP SP1, Server 2003, NetMeeting, Windows 98, and Windows ME, allows remote attackers to execute arbitrary code by causing long debug entries to be generated for the DCPROMO.LOG log file, as exploited by the Sasser worm.

Table 15 Dshield Port Report: Port 1025 Vulnerabilities[45]

Table 16 and Table 17 list the registered services and known vulnerabilities for the 3rd top targeted port: ***135***. Further correlation with information in the Scans Log is again necessary to determine possible compromised hosts and will be discussed in the ***Scans Log*** section.

Protocol	Service	Name
tcp	epmap	DCE endpoint resolution
tcp	loc-srv	NCS local location broker
udp	epmap	DCE endpoint resolution
udp	loc-srv	Location Service

Table 16 Dshield Port Report: Port 135 Services[46]

CVE ID	Pro-tocol	Src Port	Target port	Description
CAN-2003-0715	tcp	any	135	Heap-based buffer overflow in the Distributed Component Object Model (DCOM) interface in the RPCSS Service allows remote attackers to execute arbitrary code via a malformed DCERPC DCOM object activation request packet with modified length fields, a different vulnerability than CAN-2003-0352 (Blaster/Nachi) and CAN-2003-0528.

[45] http://www.dshield.org/port_report.php?port=1025

[46] http://www.dshield.org/port_report.php?port=135

CAN-2003-0605	tcp	any	135	The RPC DCOM interface in Windows 2000 SP3 and SP4 allows remote attackers to cause a denial of service (crash), and local attackers to use the DoS to hijack the epmapper pipe to gain privileges, via certain messages to the __RemoteGetClassObject interface that cause a NULL pointer to be passed to the PerformScmStage function.
CAN-2003-0533	udp	any	135	Buffer overflow in certain Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows NT 4.0 SP6a, 2000 SP2 through SP4, XP SP1, Server 2003, NetMeeting, Windows 98, and Windows ME, allows remote attackers to execute arbitrary code by causing long debug entries to be generated for the DCPROMO.LOG log file.
CAN-2003-0528	tcp	any	135	Heap-based buffer overflow in the Distributed Component Object Model (DCOM) interface in the RPCSS Service allows remote attackers to execute arbitrary code via a malformed RPC request with a long filename parameter, a different vulnerability than CAN-2003-0352 (Blaster/Nachi) and CAN-2003-0715.
CAN-2003-0352	6	any	135	Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LovSAN and Nachi/Welchia worms.

Table 17 Dshield Port Report: Port 135 Vulnerabilities[45]

The 4th top targeted port is ***119***, which is also reported previously by Greg Bassett.[47] As showed in Table 18, this port is registered to the Network News Transfer Protocol and the ***Happy99*** trojan. Since this ***Happy99*** propagates through email/Usenet attachments utilizing a modified WSOCKS32.DLL instead of through buffer overflow exploitation, these alerts are most likely false alarms that trigger on normal ***Network News Transfer Traffic (NNTP)*** traffic, especially since ***MY.NET.24.8*** - the only destination address – resolves to ***news.MY.NET***.[48]

Protocol	Service	Name
tcp	nntp	Network News Transfer Protocol
udp	nntp	Network News Transfer Protocol

[47] http://www.giac.org/practical/GCIA/Greg_Bassett_GCIA.pdf

[48] http://www.cert.org/incident_notes/IN-99-02.html

tcp	Happy99	[trojan] Happy 99
tcp	Happy99	[trojan] Happy99

Table 18 Dshield Port Report: Port 119 Services[49]

Conclusion:

This signature has a tendency to generate a lot of false positives. Further tuning of the signature is necessary to reduce the noise.

Recommendations:

- Amongst the top targeted port summary (Table 13), further research on alerts that are related to 1025, 135, 445, and 6129 is necessary to determine possible compromised machines as these ports are well known ports related MS04-011 that includes numerous buffer overflow vulnerabilities.
- In addition, alerts destined to the ephemeral ports that are not registered to known services also need further investigation.
- Consider filtering port 80 from triggering this signature to reduce false alarms.

5. Incomplete Packet Fragments Discarded

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
5357	83	88			5357			

Unique Source Ports		Unique Destination Ports	
1		1	
Top Ports	Count	Top ports	Count
0	5357	0	5357

Table 19 . Alert Logs: Ports Summary – Incomplete Packet Fragments Discarded

Source IP	Destination IP	Count
OTHER.NET.8.73	MY.NET.153.176	3074
OTHER.NET.8.71	MY.NET.153.176	2078
OTHER.NET.8.72	MY.NET.153.176	17

Table 20 . Alert Logs: Top Participants Summary – Incomplete Packet Fragments Discarded

These alerts are triggered when the old ***defrag*** preprocessor – known to have some quite serious ***failure modes*** – is used instead of the newer ***frag2*** preprocessor that was introduced in Snort 1.8.[50]. Furthermore, these alerts are mostly originated from 3 IP addresses at ***other.edu*** (***media-wm-[1|2|3].cac.OTHER.NET***) to a single IP address (***libstkpc30.libpub.MY.EDU***) on ***MY.NET*** network as showed on Table 20.

[49] http://www.dshield.org/port_report.php?port=119
[50] http://www.mcabee.org/lists/snort-users/Nov-01/msg00820.html

Deducing from their hostnames, I suspect that these are false alarms generated by from media streaming traffic which usually created a lot of fragmentations. An observation by Mayur Palankar demonstrates that "***streaming data (53%) [MS Media Player 52%] and tunneled traffic are the dominant cause of IP packet fragmentation.***"[51] In addition, there are 63 SYN scan alerts directed to MY.NET.153.76 within the 5 days period, but not of them are coming from these same source addresses.

Conclusion:
The old defrag preprocessor is still being used while its replacement – ***frag2*** – that is more memory efficient has been introduced since Snort 1.8.

Recommendations:
- Replace the ***defrag*** preprocessor with ***frag2***.

6. SMB Name Wildcard

Total	External Src IP (Unique)	Internal Dst IP (Unique)	Internal Src IP (Unique)	External Dst IP (Unique)	In-bound	Out-bound	I to I	E to E
5164			156	613		5164		

Unique Source Ports		Unique Destination Ports	
15		1	
Top Ports	Count	Top ports	Count
137	4192	137	5164
1055	184		
1061	155		
1083	142		
1093	91		

Table 21 . Alert Logs: Top Ports Summary – SMB Name Wildcard

This signature is closely related to ***archNIDS***'s ***IDS177 (NetBIOS-Name-Query)***[52] that looks for "***standard NetBIOS name table retrieval"***[53] queries based on a known IP address that can be easily generated by "***NBTSTAT –A <target ip>***" command. One noticeable difference is while IDS5177 alerts on traffic sourced from $EXTERNAL network and destines to $INTERNAL network, this signature, particularly in these 5 days ***Alerts*** logs, is triggered by outgoing traffic (from $INTERNAL to $EXTERNAL networks). Since some of the previous practicals reported this signature does also alert on incoming traffic (from $EXTERNAL to $INTERNAL)[54,47], I suspect that this ***SMB Name Wildcard*** signature has been customized to alert on any UDP traffic destined to port 137.

[51] http://www.cs.nmsu.edu/~amiya/cs584/slides/mayur.pdf
[52] http://archives.neohapsis.com/archives/snort/2000-01/0222.html
[53] http://www.whitehats.com/info/IDS177
[54] http://www.giac.org/practical/GCIA/Ian_Eaton_GCIA.pdf

Though the legitimate and malicious ***nbtstat*** requests are indistinguishable, there are usually no real reasons to have this traffic crossing the network perimeter especially when a ***Virtual Private Network/ VPN*** solution is available for remote users. Table 22 and Table 23 include the services and vulnerabilities related to UDP 137.

Protocol	Service	Name
udp	netbios-ns	NETBIOS Name Service
udp	Msinit	[trojan] Msinit

Table 22 Dshield Port Report: Port 137/UDP Services[55]

CVE ID	Pro-tocol	Src Port	Target port	Description
CVE-2001-1162	udp	any	137	Directory traversal vulnerability in the %m macro in the smb.conf configuration file in Samba before 2.2.0a allows remote attackers to overwrite certain files via a .. in a NETBIOS name, which is used as the name for a .log file.
CVE-2000-0347	udp	any	137	Windows 95 and Windows 98 allow a remote attacker to cause a denial of service via a NetBIOS session request packet with a NULL source name.
CAN-2003-0533	udp	any	137	Buffer overflow in certain Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows NT 4.0 SP6a, 2000 SP2 through SP4, XP SP1, Server 2003, NetMeeting, Windows 98, and Windows ME, allows remote attackers to execute arbitrary code by causing long debug entries to be generated for the DCPROMO.LOG log file.

Table 23 Dshield Port Report: Port 137/UDP Vulnerabilities[55]

Source IP	Source Port	Destination IP	Dest. Port	Count
MY.NET.11.7	137	169.254.25.129	137	1654
MY.NET.5.34	137	199.239.137.216	137	136
MY.NET.29.30	137	199.239.137.216	137	135
MY.NET.111.228	137	209.2.144.10	137	117
MY.NET.153.85	137	216.145.5.196	137	59
MY.NET.75.13	137	216.74.144.15	137	34

Table 24 . Alert Logs: Top Participants Summary – SMB Name Wildcard

[55] http://www.dshield.org/port_report.php?port=137

Amongst the top participants in Table 24, the most targeted IP is ***169.254.25.129***, which falls within the "***link local***" block (***169.254.0.0/16***) as quoted from RFC3330 below:[56]

"169.254.0.0/16 - This is the "link local" block. It is allocated for communication between hosts on a single link. Hosts obtain these addresses by auto-configuration, such as when a DHCP server may not be found."

When searching for the all ***SMB Name Wildcard*** alerts targeted to the ***link local block*** (***169.254.0.0/16*** instead of ***169.254.25.129***), there is a total of 2510 alerts with 4 distinct destination addresses. Both the source and destination ports are ***137*** with ***MY.NET.11.7*** as the main source addresses (1654 of 2510). As reported by Patrik Sternudd[57], MY.NET.11.7 might be a Windows domain controller, especially when nslookup resolves it to ***dc2.ad.***MY.NET. Therefore, having NetBIOS name session traffic coming out from this host can be considered normal. However, these destination addresses that are within the ***link local block*** do raise the need for further investigation.

Destination IP	Count
169.254.25.129	1654
169.254.45.176	853
169.254.138.208	2
169.254.90.17	1

Table 25 . Alert Logs: 'Local Link' Destination IPs – SMB Name Wildcard

Furthermore, there are two on-campus source addresses that trigger this signature with source ports range from ***1052*** to ***1119*** and 257 unique destination addresses. Considering the similarity of their behavior with the one described by Ken[58], these source addresses are most likely infected with the ***Opaserv*** worm[59].

Source IP	Count
MY.NET.150.44	535
MY.NET.150.198	437

Table 26 . Alert Logs: Source IPs w/ srcport != 137 – SMB Name Wildcard

Conclusion:

This signature generally creates a lot of false positive especially when both the traffic is sourced and destined from internal network or when the traffic is sourced from inside to outside network. However, it sometimes can be useful for detecting mis-configured systems or possible infected machines.

Recommendations:

[56] http://www.faqs.org/rfcs/rfc3330.html
[57] http://www.giac.org/practical/GCIA/Patrik_Sternudd_GCIA.pdf
[58] http://isc.sans.org//show_comment.php?id=85
[59] http://vil.nai.com/vil/content/v_99729.htm

- Further investigation on:
  - Alerts sourced from ***MY.NET.11.7*** and destined to the local link addresses (***169.254.0.0/16***).
  - Possible ***Opaserv*** worm infected machines: ***MY.NET.150.44*** & ***MY.NET.150.198***.
- Tune the signature, below are several alternatives:
  - Alert only on traffic that comes from $EXTERNAL to $INTERNAL network. Other compensating signatures can be used to detect worm specific activities.
  - When choosing not to alert only on incoming traffic from $EXTERNAL network, modify the signature to fire when the source port is not 137.
- Consider modifying the egress and ingress filtering when they are not currently in place:
  - Block outgoing and incoming Windows NetBIOS ports (UDP and TCP 135-139 and 445) at the border firewall/ router
  - Block traffic destined the '***local link***' IP ranges from leaving the network perimeter.

Scans Logs

Scan Type Summary		Top 10 Destination Ports			
		From Internal		From External	
Scan Type	**Count**	Port	Count	Port	Count
SYN	16,976,176	135	5863846		
UDP	5,568,793	445	5625138		
FIN	27,700	53	4536988		
INVALIDACK	3,671	2745	1013177		
UNKNOWN	2,205	1025	682935		
NULL	898	80	597235		
NOACK	874	3127	494037		
VECNA	271	6129	419539		
XMAS	59	139	326591		
SPAU	50	25	311070		
FULLXMAS	44				
SYNFIN	29				
NMAPID	24				

Table 27 . Scans Logs: Scan Alerts Summary

Source IP	**Alert Count**	**Unique DstIP Count**	**Unique DstPort Count**	**Top DstPort Value**
MY.NET.190.92	10198630	2414793	128	135(50%),445(49.7%),5000,139,6667,161,137,53,80,8080
MY.NET.111.51	3895364	565546	31	2745(21.7%),135(17.7%),1025(14.6%),445(12.5%), 3127(10.6%),6129(9%),139(7.5%),80(6.3%),411,6666
MY.NET.1.3	3811608	123024	1,916	53(99.6%),123,10123,45190,1170,60008,60238,60261,18341,60369

MY.NET.1.4	752087	54786	748	53(98.6%),123,45197,1170,10123,60008,60261,60238,60033,18330
MY.NET.84.235	472085	141190	13,852	4672(45%),4673(14%),4662(10%),4665,4246,80,5672,4671,4661,21065
MY.NET.34.14	215360	3260	2	25(98%),113
MY.NET.110.72	203517	17483	9,833	32785,32794,32777,12109,32836,1938,32770,12108,32776 3014
MY.NET.153.174	188460	17850	10	2745(16%),135(15%),1025(13%),445(12%),3127(11%),6129(10.8%),139(10%),80(9%),21,29033
MY.NET.97.108	133676	71698	7	2745(39.5%),1025(23.3%),3127(15.5%),6129(12.7%),80(8.8%),6666,443
MY.NET.97.103	87731	37836	30	80(89%),2745(3%),1025(2.7%),3127(2.2%),6129(2.1%),3531,2626,2090,2266,1263

Table 28 . Scans Logs: Top 10 Attackers - Internal

Source IP	Alert Count	Unique DstIP Count	Unique DstPort Count	Top DstPort Value
213.180.193.68	71034	2: MY.NET.25.10 MY.NET.25.68	61446	47203,9765,36448,35097,7237,59162,31332 ,29173,28380,3757
210.139.118.246	59315	2: MY.NET.1.3 MY.NET.190.92	48773	5368,19084,11117,31852,35441,32342,26568,1181,63001,3777
67.31.152.200	59146	41	1328	80,731,1497,82,373,2008,344,510,382,1815
66.212.217.203	53067	15683	1	17300
68.66.247.59	36051	12592	4	3128(33.4%),1080(33.3%),10080(33.2%),3127
80.203.201.148	34085	12634	1	80
211.78.176.3	30114	12896	1	6129
211.43.90.104	27949	15540	1	443
218.55.179.190	27365	15453	1	6129
68.71.57.193	27343	15473	1	4000

Table 29 . Scans Logs: Top 10 Attackers - External

Victim IP	Alert Count	Unique SrcIP Count	Unique DstPort Count	Top Destination Ports
MY.NET.25.68	71164	98	61452	6129,113,4899,4000,80,20168,36448,35097 ,59162,7237
MY.NET.190.92	63602	2269	201	135,1433,6129,4899,2803,1771,4000,20168 ,2215,4751
MY.NET.97.202	9878	620	21	6346(99.3%),6129,80,4000,20168,4751,4899,17300 ,3306,5570

MY.NET.97.15	7957	479	25	6346(99%),6129,4000,4899,80,17300,20168,21,4751,113
MY.NET.97.125	6438	595	21	6346(98.7%),6129,20168,80,4000,4751,4899,17300,1257,21
MY.NET.97.84	5983	446	22	6346(98.6%),6129,80,4000,4899,20168,21,4751,17300,1257
MY.NET.190.97	4494	176	1373	4444,135,1433,137,80,6129,139,4899,4000,20168
MY.NET.34.11	4285	107	1345	80,6129,4899,4000,21,20168,5900,710,488 ,381
MY.NET.12.6	4190	419	1229	25(66.9%),0,6129,21,80,4899,20168,4000,389 ,17300
MY.NET.97.83	3663	54	22	6346(97.8%),6129,80 ,4899,4000,21,4751,20168,17300,443

Table 30 . Scans Logs: Top 10 Victims - Internal

Phatbot/Agobot Worm propagation attempts

Amongst the top on-campus '***scanners***', there are five hosts (***MY.NET.190.92, MY.NET.111.51, MY.NET.153.174, MY.NET.97.108, MY.NET.97.103***) that demonstrate symptoms of ***Agobot/Gaobot***[60]. Variants of this worm are known to perform scans on port:[61]

- ✓ *"135 for MS03-039 "DCOM2" vulnerability*
- ✓ *139 for MS03-049 Workstation vulnerability*
- ✓ *1433 for weak MSSQL administrator passwords*
- ✓ *2082 for CPanel vulnerability (OSVDB ID: 4205)*
- ✓ *2745 for backdoor left by the Bagle Virus*
- ✓ *3127 for MyDoom.A backdoor*
- ✓ *5000 for MS01-059 UPnP vulnerability*
- ✓ *6129 for Dameware vulnerability (OSVDB ID: 3042)*
- ✓ *80 for MS03-007 WebDav vulnerability*
- ✓ *135, 445 and 1025 for MS03-032 vulnerability*
- ✓ *139 and 445 for weak NetBIOS passwords"*

On-campus '***scanners***' that scan on at least two of the above ports can be found on Table 31.

Source IP	Unique Port Count
MY.NET.153.174	80, 135, 139, 445, 1025, 2745, 3127, 6129
MY.NET.153.195	80, 135, 139, 445, 1025, 2745, 3127, 6129
MY.NET.111.51	80, 135, 139, 445, 1025, 2745, 3127, 6129
MY.NET.97.169	80, 1025, 2745, 3127, 6129
MY.NET.97.25	80, 1025, 2745, 3127, 6129
MY.NET.98.65	80, 1025, 2745, 3127, 6129

[60] http://www.lurhq.com/phatbot.html
[61] http://seclists.org/lists/incidents/2004/Apr/0063.html

MY.NET.97.12	80, 1025, 2745, 3127, 6129
MY.NET.97.57	80, 1025, 2745, 3127, 6129
MY.NET.97.124	80, 1025, 2745, 3127, 6129
MY.NET.97.103	80, 1025, 2745, 3127, 6129
MY.NET.97.43	80, 1025, 2745, 3127, 6129
MY.NET.97.30	80, 1025, 2745, 3127, 6129
MY.NET.97.159	80, 1025, 2745, 3127, 6129
MY.NET.97.126	80, 1025, 2745, 3127, 6129
MY.NET.97.78	80, 1025, 2745, 3127, 6129
MY.NET.97.49	80, 1025, 2745, 3127, 6129
MY.NET.97.235	80, 1025, 2745, 3127, 6129
MY.NET.97.92	80, 1025, 2745, 3127, 6129
MY.NET.190.92	80, 1025, 2745, 3127, 6129
MY.NET.97.129	80, 1025, 2745, 3127, 6129
MY.NET.97.108	80, 1025, 2745, 3127, 6129
MY.NET.98.53	80, 1025, 2745, 3127, 6129
MY.NET.81.59	80, 135, 445
MY.NET.84.235	80, 135, 1025, 1433, 2745, 5000
MY.NET.153.90	80, 1433, 2082, 2745, 3127
MY.NET.70.96	80, 135
MY.NET.71.235	80, 445
MY.NET.153.99	80, 139, 5000

Table 31 . Scans Logs: Possible Agobot Infected Hosts- Internal

Recommendations:

1. Disconnect the systems listed on Table 31, especially those that are scanning on more than 3 different Agobot related ports and have the systems owner perform virus scanning and disinfections. The ***MY.NET.97.0/24*** subnet seems to be highly infected.
2. Require the following on each Microsoft Windows systems connected to the campus network through a network access control mechanism:
   - ✓ Enable automatic windows updates or have other means in ensure that machines are keep current on their security patches
   - ✓ Require strong user passwords and disable null sessions / anonymous logons
   - ✓ Require up-to-date anti-virus software
3. Have policy and procedure in place to disconnect any infected systems from the network as soon as possible
4. Consider modifying the egress and ingress filtering to block outgoing and incoming Windows NetBIOS ports (UDP and TCP 135-139 and 445) at the border firewall/ router.

P2P applications

Five of the hosts (grey shaded) listed in the Top Internal Victims on Table 30 are involved in Gnutella Peer to Peer file sharing network as they are consistently

scanned by other ***Gnutella servents*** on port ***6346***. In addition, ***MY.NET.84.235*** (from Table 28) seems to be using the eDonkey (TCP 4662) and eMule (UDP 4672) P2P programs.

Recommendations:
Due to the amount of traffic that can be consumed and the increased risks (from both security and legal perspective) associated with P2P applications, consider reviewing the current policy regarding P2P and the use of packet shaper devices to control the amount of P2P traffic.

Other Scans Activities

1. Two of ***MY.NET's*** DNS servers (***MY.NET.1.3 and MY.NET.1.4***) appear in Table 28 as the top talkers for port scanning that are sourced on-campus. These are mostly like false alarms and it's recommended to add the DNS server addresses to the Snort's ***portscan-ignorehosts*** variable[62] to reduce these noises.

2. There are several machines off-campus that are scanning for ***MY.NET*** addresses space for specific ports such as ***80*** (***http***), ***6129*** (***Dameware***), ***443*** (***https***), ***4000***[63], ***17300*** (***Kuang2TheVirus***). Checking the Alert logs, there are 2 on-campus machines (***MY.NET.150.44*** and ***MY.NET.150.198***) that consistently respond to these probes indicated by the name queries alerts from the two on-campus addresses to these off-campus addresses generated almost the same timestamps are the originating probes. Further investigation is needed on these two machines to determine the real impact, especially because they are also suspects for Opaserv worm infection in the ***SMB Name Wildcard*** alerts section.

   In addition, these two hosts are also reported by Peter Storm in his practical as responding to proxy scans. He also noted that "*personal firewalls, server log tools, and similar tools may respond with SMB name queries*".[34] A link diagram on describes the relationships among these hosts. I

Attacker IP	Destination Port
66.212.217.203	17300
80.203.201.148	80
211.78.176.3	6129
211.43.90.104	443
218.55.179.190	6129
68.71.57.193	4000

[62] http://lists.jammed.com/incidents/2001/05/0239.html
[63] http://www.dshield.org/port_report.php?port=4000

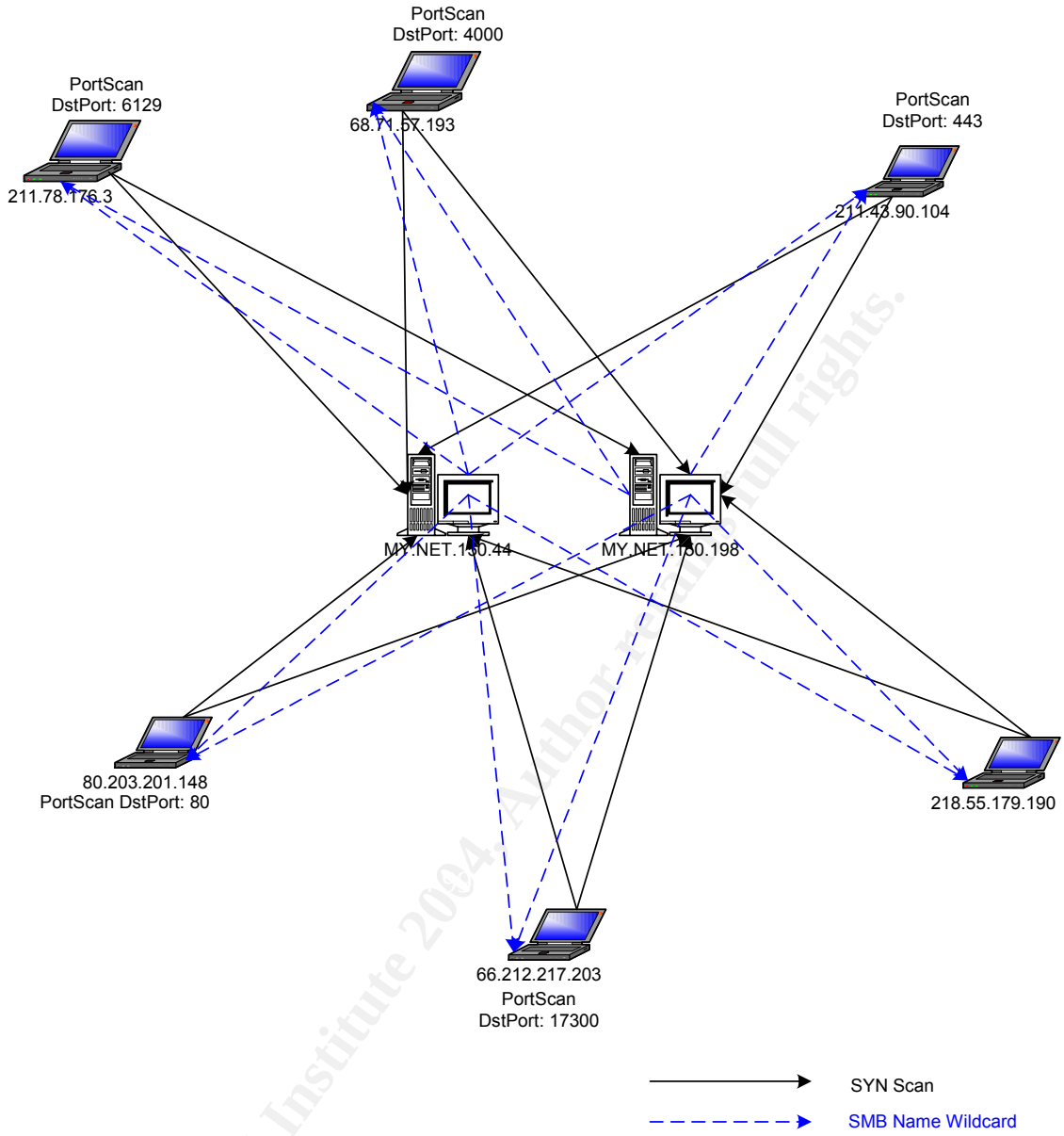


Figure 6. Link Diagram – Port Scans to MY.NET.150.44 & MY.NET.150.198

3. One of the top external attackers (***68.66.247.59)*** seems to be targeting the proxy ports, i.e. ***3128*** (***squid proxy***), ***1080*** (***socks proxy***) and ***10080*** (***Amanda*** – open-source backup).
4. There are two scanning activities that appears to target specific on-campus address:
   - ✓ ***213.180.193.68*** - SYN scanning ***MY.NET.25.10*** and ***MY.NET.25.68***
   - ✓ ***210.139.118.246*** – SYN scanning ***MY.NET.1.3*** and ***MY.NET.190.92***

   The owners of these on-campus machines should be made aware of these probing and possible compromised when appropriate host-based security is not in place.

OOS Logs

Source IP	Unique Dst IP Count	Alerts Count
68.54.84.49	1	1115
66.225.198.20	1	130
4.62.160.223	1	111
68.48.163.221	3	98
203.172.97.150	6	90
4.13.172.39	3	89
67.114.19.186	1	72
68.7.123.221	1	65
68.121.194.43	1	56
207.87.144.68	2	53

Table 32 . OOS Logs: Top 10 Talkers

TCP Flags	Count
12****S*	2976
********	520
U***	33
**U*P*SF	22
****P***	21
12UAPRSF	19
*2U*PRSF	19
*2UA**SF	13
**U*PRSF	13
12UAPR*F	11

Table 33 . OOS Logs: Top 10 TCP Flags

As mentioned by Peter Storm in his practical[34], these Out of Specification (OOS) alerts are generated by Snort when TCP options or flags anomalies are detected. Based on Table 33, 74.5% of the alerts in ***OOS Logs*** has are SYN packets with both TCP reserved bits set. Currently, these reserved bits are known as ECN bits and thus, they can be set only when the Explicit Congestion Notification (ECN) protocol – RFC 3168 – is employed. Since different operating systems respond differently to packets that have these reserved bits, tools such as ***Queso*** and ***nmap*** utilize this 'feature' to perform operating system finger printing. [64]

CWR	ECN echo	URG	ACK	PSH	RST	SYN	FIN

In addition, ***13%*** of the ***OOS*** alerts are related to ***null TCP*** packets, i.e. ***TCP packets*** without any ***TCP flags***. The rest of the alerts – 12.5% – are spread among the other 111 different TCP flags combinations.

[64] GCIA Material – Part 3.2 and 3.3, page 5-19

OOS Top Talker - 68.54.84.49

The top talker in the ***OOS Log*** – ***68.54.84.49*** – appears to target ***MY.NET.6.7*** on port 110 (***pop3***) with ***1115 SYN*** packets that have both ***TCP reserved*** bits set.

```
68.54.84.49: pcp01741335pcs.howard01.md.comcast.net

03/31-00:21:59.910595 68.54.84.49:35362 -> MY.NET.6.7:110
TCP TTL:51 TOS:0x0 ID:18221 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x46CD73CD  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 811545824 0 NOP WS: 0
```

In addition, there are ***3259*** alerts found in the ***Scans Log*** sourced from ***68.54.84.49*** to ***MY.NET.6.7*** on the same port (***pop3***), which is a legitimate service provided by ***MY.NET.6.7*** and is currently using ***pop3*** server ***v2001.78***. While there is not enough information to conclude these ***OOS*** and ***Scans*** as malicious, it is recommended to modify the pop3 login banner to exclude the software version.

OOS Null TCP Packet

Source IP	Count
203.172.97.150	90
68.121.194.43	56
68.7.123.221	43
4.62.160.223	38
4.13.172.39	24
MY.NET.70.37	20
68.164.89.241	15
68.5.204.185	12
165.134.62.223	11
165.134.48.220	11

Table 34 . OOS Logs: Top 10 Sources – Null TCP Packets

Top Attacker - 203.172.97.150

```
03/31-02:57:13.005315 203.172.97.150:113 -> MY.NET.25.68:54439
TCP TTL:179 TOS:0x0 ID:34611 IpLen:20 DgmLen:40
******** Seq: 0x6228F20A  Ack: 0xDBAAD32F  Win: 0x0  TcpLen: 20
```

All of the null packets from ***203.172.97.150*** are sourced from port ***113*** and directed to 6 different on-campus addresses within ***MY.NET.25.0/24*** subnet, i.e.:

Source Port	Destination IP	Destination Port	Count
113	MY.NET.25.67	38094	2
113	MY.NET.25.67	41463	3
113	MY.NET.25.67	42207	3
113	MY.NET.25.67	42544	3
113	MY.NET.25.68	34811	4

113	MY.NET.25.68	35221	3
113	MY.NET.25.68	54439	7
113	MY.NET.25.68	60802	3
113	MY.NET.25.68	63442	3
113	MY.NET.25.68	64339	5
113	MY.NET.25.69	37404	2
113	MY.NET.25.69	53488	3
113	MY.NET.25.69	54984	7
113	MY.NET.25.69	55395	2
113	MY.NET.25.70	39209	5
113	MY.NET.25.70	40863	3
113	MY.NET.25.70	41599	2
113	MY.NET.25.71	53535	3
113	MY.NET.25.71	55303	6
113	MY.NET.25.73	33067	7
113	MY.NET.25.73	41822	2
113	MY.NET.25.73	60573	3
113	MY.NET.25.73	65438	9

Table 35 . OOS Logs: Destination Addresses – Null TCP Packets from 203.172.97.150

Both the ***Scans and Alerts Logs*** also show similar information there are null scans coming from ***203.172.97.150*** to various machines on ***MY.NET.25.0/24*** subnet, i.e. hosts in Table 35 and ***MY.NET.25.66***.

Looking at both the ***Alerts and Scans Logs*** for interesting events (Table 36) that come from these 6 on-campus addresses, I notice the following that they are also the source addresses for:

- ✓ ***186 High port 65535 tcp - possible Red Worm – traffic*** alerts with two unique destination ports, i.e.: ***25***[65]and ***113***[66].
- ✓ ***118191 SYN*** scans alerts that are also destined to port ***25*** and ***113***.

	Alerts Logs		Scans Logs	
Source IP	SrcPort	Count	DstPort	Count
MY.NET.25.66	65535	6	25 113	8102 2367
MY.NET.25.67	65535	44	25 113	10080 2627
MY.NET.25.68	65535	19	25 113	5082 1714
MY.NET.25.69	65535	21	25 113	21194 7694
MY.NET.25.70	65535	37	25 113	21733 7773
MY.NET.25.71	65535	30	25 113	15514 5921
MY.NET.25.73	65535	29	25 113	6533 1857

Table 36 . Alerts Logs: Alerts from MY.NET.25.66-73

[65] http://www.dshield.org/port_report.php?port=113
[66] http://www.dshield.org/port_report.php?port=25

While these 6 addresses appear to be ***MY.NET's*** mail servers that are ***Sendmail 8.13.1*** (the latest version that is available since 2004-07-31), it is still recommended to review the mail servers' syslog files to ensure the system-level integrity.

The following are some quite old links related to ***sendmail*** – ***Identd*** attacks:

- ✓ Posting by Guido Stevens: http://archives.neohapsis.com/archives/incidents/2000-04/0008.html
- ✓ CVE-1999-0204: Sendmail 8.6.9 allows remote attackers to execute root commands, using ident.
- ✓ CVE-1999-0204: Sendmail 8.6.9 allows remote attackers to execute root commands, using ident.

Other analysis

Top External Attackers

67.31.152.200

Signature	Count
MY.NET.30.3 activity	3560
MY.NET.30.4 activity	2967
TFTP - External TCP connection to internal tftp server	58

Table 37. Alert Summary – 2nd Top Off-campus Attacker IP: 67.31.152.200

This section focuses only on the last signature alerts because the first two have been discussed in earlier sections. ***67.31.152.200*** targets ***32*** different on-campus hosts on the "***TFTP - External TCP connection to internal tftp server***" signature; each with a maximum of 3 attempts. This signature is also a custom signature; it seems to look for traffic that either sourced from or destined to TCP port 69. This is concluded from the existence of 32 other ports in addition to port 69 (default TFTP port) for this signature alerts as showed in Table 38. Based on the records from the ***Scans*** log, all records sourced from ***67.31.152.200*** only have the SYN flag set, and thus these are most likely SYN scans for port 69.

Alerts Count	TCP 69	TCP !69
As source port	76	95
As destination port	95	76

Table 38. Ports Summary: External TCP connection to internal tftp server

From these 32 on-campus victim addresses, there are 22 systems – including ***MY.NET.30.3*** and ***MY.NET.30.4*** – that responded to ***67.31.152.200***'s SYN scans. Since the Alert log does not provide any information regarding the TCP flags sets on these response packets, it's hard to determine the ones what do listen on ***TCP 69***. Assuming that there is no alert/record loss during the data clean up process (which is unlikely), it's possible to guess by comparing the number of SYN scans sent by ***67.31.152.200*** to the victims (**S**), the number of ***TFTP - External TCP connection to internal tftp server*** alerts sourced from ***67.31.152.200*** (**A**) and destined to it from the victims (**V**). I can assume that none of the targeted systems are vulnerable when the following condition is met.

```
A == S and (V == S or V == 0)
```

This means that only SYN scan packets ever sent to the victims and the victims either respond with RST packet or do not reply at all. However, the attacker might still be able to use this information for OS/ network fingerprinting.

Recommendations:

1. Although ***Trivial File Transfer Protocol/ TFTP*** is not a secure file transfer protocol, it remains a popular service used to transfer configuration files and

software updates to and from network devices. Therefore, appropriate access controls should be put in place to secure them when being used as to keep the configurations of network and security devices.
2. Restrict access for TFTP server of network devices from certain places. Remote access should require VPN.
3. Periodically audit the network for unnecessary TFTP servers.

Registration Information of External Source Addresses

These addresses are chosen as they showed up as the Top External Scanners that appear to target on certain on-campus hosts or certain trojan/backdoor services.

External IP	Registration Info	Contact Info
213.180.193.68 proxychecker.yandex.net	inetnum: 213.180.192.0 - 213.180.193.255 netname: COMPTEK-NET1 descr: CompTek International/Yandex LLC descr: 3, Gubkina str., Moscow, 117809 country: RU	Contact: Yandex LLC Network Operations Address: Yandex LLC 40A Vavilova st. 117333, Moscow, Russia Phone: +7 095 9743555 Fax-no: +7 095 9743565 E-mail: noc@yandex.net
210.139.118.246 pl502.nas922.n-yokohama.nttpc.ne.jp	inetnum: 210.139.0.0 - 210.139.127.0 netname: INFOSPHERE descr: InfoSphere descr: NTTPC Communications, Inc. country: JP	Administrative Contact: HH1558JP Technical Contact: RK448JP Technical Contact: HK8557JP E-mail: ip@sphere.ad.jp tech-contact@sphere.ad.jp staff@db.nic.ad.jp
66.212.217.203 dhcp-66-212-217-203.myeastern.com	inetnum: 66.212.192.0 - 66.212.223.255 Inetnum: 66.212.216.0 - 66.212.219.255 CustName: myeastern.com Address: 61 Myrock Ave Address: Waterford, CT 06385 City: Plainfield StateProv: CT PostalCode: 06374 Country: US	Tech Contact: OH46-ARIN O'Brien, Hugh hughobrien@myeastern.com OrgAbuseHandle ABUSE148-ARIN Abuse@myeastern.com OrgTechHandle OTP-ARIN Parsons, Owen T. owenparsons@myeastern.com Phone: +1-860-442-5616
211.78.176.3 adsl-211-78-176-3.HCON.sparqnet.net	inetnum: 211.78.160.0 - 211.78.191.255 netname: NCICNET-TW descr: New Centry InfoComm Tech. Co., Ltd. descr: 12F, No. 468, Rueguang Rd. Taipei descr: Taiwan 114 country: TW	Contact: Claire Chang PC Home 8Fl., No. 378, Fushing N. Rd., Jungshan Chiu Taipei Taiwan, TW Phone: +886-2-7700-8888 E-mail: clairechang@ncic.com.tw
68.71.57.193	inetnum: 68.71.48.0 - 68.71.63.255	Tech Contact:

	descry: Adelphia Address: 1 North Main Street City: Coudersport StateProv: PA PostalCode: 16915 Country: US RegDate: 2002-10-22 Updated: 2002-10-22	AH102-ARIN Adelphia Hostmaster +1-814-274-0638 ipadmin@adelphia.net OrgAbuseHandle: IPE-ARIN Internet Policy Enforcement +1-866-473-2909 abuse@adelphia.net OrgTechHandle: CKI8-ARIN Kio, Carolyn +1-888-512-5111

Table 39 . Scans Logs: Top 10 Attackers - External

Analysis Process:

1. Combine the 15 log files (5 of each type) into three files (1 of each type).
2. Experiment with data analysis tools such as SnortSnorf and Snortalog to process the data. Unfortunately, the log files seems to be too large and contains numerous invalid records.
3. Decide to use the parsing and database creation scripts obtained from the practical reports of Samuel Adams[67] and Les Gordon[68] after some customizations
4. Clean up the data, especially the scans logs. This can take quite some time.
5. Create the database (mysql) and upload the data.
6. Add indexes to the scans log table to speed up the query processing
7. Use mysqldump to backup the whole database. This is particularly important considering the amount of time that has been spent to upload and clean up the data.
8. Start analyzing the data using SQL queries.

Reference (Part 2 & 3)

http://isc.sans.org/logs/Raw/README
http://tcpreplay.sourceforge.net/pcapmerge.html
http://www.sourceforge.net/projects/tcpreplay/
http://tcpdump.org/tcpdump_man.html
http://standards.ieee.org/regauth/oui/oui.txt
http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html
http://www.snort.org/dl/contrib/data_analysis/snortsnarf/
http://www.snort.org/snort-db/?sid=522
http://www.geektools.com/whois.php
http://www.securityfocus.com/infocus/1577
http://www.securiteam.com/windowsntfocus/Patch_Available_for_the__IP_Fragment_Reassembly__Vulnerability.html
http://www.iana.org/assignments/port-numbers

[67] http://www.giac.org/practical/GCIA/Samuel_Adams_GCIA.pdf
[68] http://www.giac.org/practical/GCIA/Samuel_Adams_GCIA.pdf

http://rfc-gnutella.sourceforge.net/developer/share/intro.html#Background
http://www.faqs.org/rfcs/rfc793.html
http://www.stearns.org/p0f/p0fr.fp
http://www.mynetwatchman.com/ListIncidentsbyIP.asp
http://www.inet-sec.org/docs/DoS/fragma.html
http://www.cisco.com/warp/public/770/nifrag.shtml
http://www.snort.org/snort-db/sid.html?sid=499
http://www.insecure.org/sploits/ping-o-death.html
https://www.sans.org/resources/malwarefaq/stacheldraht.php
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0128
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0345
http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00062.html
http://www.kb.cert.org/vuls/id/150227
http://www.uoregon.edu/~joe/proxies/open-proxy-problem.pdf
www.lurhq.com/zindos.html
http://news.com.com/Google%2C+other+engines+hit+by+worm+variant/2100-1023_3-5283750.html
http://support.novell.com/cgi-bin/search/searchtid.cgi?/10065719.htm
http://www.giac.org/practical/GCIA/Michael_Meacle_GCIA.pdf
http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf
http://www.leu.bw.schule.de/netze/novell/ml2/patches/nw6sp3.txt
http://www.f-secure.com/v-descs/adore.shtml
http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf
http://www.iana.org/assignments/port-numbers
http://securityresponse.symantec.com/avcenter/venc/data/w32.korgo.f.html
http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=39437
http://www.sagonet.com/servers/gaming.php
http://www.snort.org/snort-db/sid.html?sid=648
http://www.insecure.org/stf/smashstack.txt
http://isc.sans.org/diary.php?date=2004-04-30
http://www.dshield.org/port_report.php?port=1025
http://www.dshield.org/port_report.php?port=135
http://www.giac.org/practical/GCIA/Greg_Bassett_GCIA.pdf
http://www.cert.org/incident_notes/IN-99-02.html
http://www.dshield.org/port_report.php?port=119
http://www.mcabee.org/lists/snort-users/Nov-01/msg00820.html
http://www.cs.nmsu.edu/~amiya/cs584/slides/mayur.pdf
http://archives.neohapsis.com/archives/snort/2000-01/0222.html
http://www.whitehats.com/info/IDS177
http://www.giac.org/practical/GCIA/Ian_Eaton_GCIA.pdf
http://www.dshield.org/port_report.php?port=137
http://www.faqs.org/rfcs/rfc3330.html
http://www.giac.org/practical/GCIA/Patrik_Sternudd_GCIA.pdf
http://isc.sans.org//show_comment.php?id=85
http://vil.nai.com/vil/content/v_99729.htm
http://www.lurhq.com/phatbot.html
http://seclists.org/lists/incidents/2004/Apr/0063.html
http://lists.jammed.com/incidents/2001/05/0239.html
http://www.dshield.org/port_report.php?port=4000

http://www.dshield.org/port_report.php?port=113
http://www.dshield.org/port_report.php?port=25
http://www.giac.org/practical/GCIA/Samuel_Adams_GCIA.pdf

Last Updated: August 17th, 2017

Upcoming Training

SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Baltimore Fall 2017 - SEC503: Intrusion Detection In-Depth	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Scottsdale SEC503**	Scottsdale, AZ	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Ottawa SEC503	Ottawa, ON	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
Community SANS Pensacola SEC503	Pensacola, FL	Nov 27, 2017 - Dec 02, 2017	Community SANS
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Munich December 2017	Munich, Germany	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Northern VA Spring - Tysons 2018	Tysons, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced