Global Information Assurance Certification Paper

Intrusion Detection In Depth
GCIA Practical Assignment

Version 3.3

Submitted by
ASHLEY THOMAS
March 17, 2003

Table of Contents

Summary:
The document is divided into three parts. Part 1 (Describe the state of intrusion detection) talks about challenges and approaches for Intrusion detection in a high speed network. Part 2 covers analysis of three network detects. Two of them were posted on the intrusions@incidents.org mailing list for questions or comments from experts and the answers are included. In Part 3, a network audit is performed based on log files downloaded from incidents.org mailing list.

Part 1: Intrusion detection in High-speed networks

Abstract:
The Intrusion detection systems (IDS) have been trying to catch up with the ever-increasing network speed. The network and Link bandwidth is increasing at a very high rate [1]. Researchers all over have been trying to increase the performance of IDS to be able to monitor traffic greater than 100 Mbps or even Gigabit traffic. This paper discusses the issues that the current IDS technology is facing with respect to monitoring High-speed networks and their probable solutions.

Introduction:
IDS are used to detect abnormal operations (intrusions) that might occur in the network due to a malicious hacker or due to erroneous applications resulting in disruption of network operations. There exist different classifications of IDS, based on different criteria. One important classification is based on how the audit data is collected. There exist two types of IDS categories - Host-based IDS, also known as the HIDS and Network-based IDS a.k.a NIDS. The audit data for an HIDS is collected from the log files on the host machines on which it operates where as the audit data for the NIDS is from the network traffic. This document deals with the issues regarding an NIDS.
Another classification is based on the type of analysis that an IDS performs on the audit data. This classifies an IDS into Signature based, Anomaly based, Protocol Analysis based etc. Often an IDS does a mixture of all the above analyses.

Basic operation of NIDS:
A Network IDS monitors all the inbound and outbound traffic to/from the network, which requires the IDS's network interface card to operate in a mode called promiscuous mode. In this mode the card accepts all the packets regardless of the destination MAC address. The placement of IDS on the network is of high importance. The IDS has to be placed at such a location where it can monitor (promiscuously) all the packets bound to the network that it is monitoring. This makes Demilitarized zone (DMZ) a good location for an NIDS. The NIDS operates in what is known as a Fail-open mode. A copy of the packet will be analyzed by the IDS while the original packet goes to its destination. This behavior of a NIDS means that even if that packet is detected to be part of an intrusion it is still going to reach its destination. On the contrary a firewall

operates in Fail-close mode (i.e. when a packet that is either bad or not allowed arrives it does not pass through the firewall).

Most of the IDS are built over a user level packet capture library called libpcap [2] or winpcap [3] (windows version of libpcap). The libpcap layer captures the packet from the wire and delivers it to the IDS. Nowadays the IDS makes the interface card go to a stealth mode; in which the interface is not assigned an ip-address and is therefore hidden from any other node on the network.

Impact of High-speed networks on NIDS:

One common design goal that Intrusion detection systems have is that of real time detection and notification [5] [4]. This means that the analysis of traffic is not done offline but at real time. In offline Intrusion detection, network traffic data in the form of TCPdump or other logs will be fed to the IDS at the end of the day or in certain intervals. This makes the detection possible only after a significant amount of time. The fact that NIDS operate in Fail-open mode makes it critical that intrusive activities be detected as soon as possible and that appropriate actions be taken. This motivates the need for real time intrusion detection systems. However real time detection and notification is a challenge since NIDS has to process all the packets that flow through the network and the computation power has not kept pace with the increases in network bandwidth. A few factors or points that are directly affected due to this are [6]:

1. **Performance**

The performance of an Intrusion detection System is the rate at which audit events are processed [6]. If performance falls below some predetermined level then the IDS will not be able to keep up with the network traffic. In short, the processing has to be 'on the fly' i.e. in the worst case the packet processing time should be less than the inter packet arrival time.

2. **Completeness**

This point refers to the functionality of the IDS, the size of the signature database etc. It is very important that the NIDS perform basic functions like IP fragmentation reassembly and TCP stream reassembly. Also the status of ths signature database is very important. It has to be updated regularly and quickly as a new attack signatures are published.

It can be noted that the above two points are difficult to attain simultaneously. When complete coverage and analysis is done, the processing time increases and performance is affected and vice versa. Completeness should not be compromised when you aim for high-speed network intrusion detection.

Consequence of low performance:

When the IDS is not able to keep up with the traffic, it starts dropping packets, resulting in missing attacks; thereby defeating the very purpose why IDS was used. One could argue that although the IDS may drop packets it need not drop packets when attack happens (i.e. miss attacks) but, the fact that the probability of missing attacks is non-zero itself is bad. Moreover the attacker knowing this weakness of IDS can plan his attack such that in the first phase of attack he

overloads the IDS to a point where it starts dropping packets and then sneak in and attack.

From reports like IDS Group tests from NSS [7] and Network Computing [8] we see that even the best of NIDS doesn't fare well when tested under performance load tests especially with small packets (64 bytes). They started dropping packets at traffic loads lower than 100Mbps. As the report from Network Computing [8] mentions not many IDSs give out information about whether they are dropping packets, which is even worse.

Bottlenecks:

Having seen that IDSs do not perform well at heavy traffic loads and miss attacks, lets try to consider what are all the issues or performance bottlenecks that the current IDSs have. Some might be general and the others specific to some IDS.

1. Processor speed

Intrusion detection on Gigabit networks would have been a piece of cake if we had processors with an infinite processing capability. The rate at which processor speed increases is lower than the rate at which networking rates are increasing [12] [1].

2. Libpcap – packet capture mechanism

Neil Desai in his paper [10] describes libpcap to be one of the main bottlenecks that the IDS (specifically Snort) is facing. As Neil Desai [Ref] points out; going for an OS specific and NIC specific libpcap would be more efficient but then portability is affected. There is a compromise involved.

The libpcap provides system independence by hiding the different raw packet capture mechanisms on different flavors of UNIX. And different flavors of Unix OSs have different packet capture mechanisms and hence different rate at which they can capture. Most of them require copying the data from kernel space to user space. Operating systems like *BSD use BPF (BSD packet filter) [Ref] which avoids this. Similarly Linux 2.4.x kernel also uses memory-mapped mechanism to overcome this extra copying.

3. String / Pattern matching algorithms

This can be a bottleneck especially when the IDS tends to be a signature based IDS [10]. Snort [5] as mentioned above is a signature based IDS. 1086 out of 1270 rules in the signature base involves some kind of string/pattern matching. For such IDS it is imperative that the string/pattern matching be as fast as to satisfy the on the processing rule.

4. Increasing signature database

As new attacks are discovered almost everyday, new signatures are added into the database. It also means that for each packet, the IDS will need to do more analysis. This again increases the load on the IDS.

5. Poor performing platforms

IDS Group tests [7] mention a specific platform, which they found to be poorly performing. According to them:

> During testing we noticed significant problems under Red Hat Linux (both 6.2 and 7.1) when using 3Com 3C905 network cards, where the driver appeared to be

overwhelmed at 100 per cent network loads thus preventing the IDS sensor from detecting attacks. This effect was not limited to any one IDS product, and even occurred when using *tcpdump* with the interface in promiscuous mode. There is clearly a problem somewhere in the chain of network card – driver – packet capture library, which were unable to resolve in the time available to us. For now, therefore, we could not recommend this combination of OS and network card as a platform for any IDS system.

6. IDS specific issues

There can be IDS specific issues that might be a bottleneck for it. We discuss two cases of open source IDS.

Case 1: Consider Bro, an open source NIDS from Berkeley labs developed by Vern Paxson [4]. The IDS has an Event engine over the libpcap layer and above the Event engine is a policy interpreter. The Event engine generates events for different network events like tcp_connection_established and the policy script has handlers for each such event. Vern mentions [4] that the interpretive overhead is indeed significant and plans to develop a compiler for the same. *At heavy loads if the interpreting cannot be done fast enough the on the fly processing requirement is not satisfied and will lead to dropped packets.*

Case 2: Consider Snort, another open source NIDS by Martin Roesch. Neil Desai discusses the main bottlenecks of Snort in his paper [Neil]. Listing them again they are:

1. Libpcap (discussed above)
2. String matching algorithm.
3. Clearing out data structures.
4. Checksum verification.

Approaches / Solutions:

This section discusses the different approaches or solutions for the above mentioned bottlenecks, which will push the IDS to higher and higher traffic loads.

1. Load balancing:

The load balancing approach tries to solve the problem of IDS in high-speed networks by balancing the load among different sensors. The division of labor can be done based on a)Protocol b) destination subnet ip address.

Different sensors can analyze traffic of different protocols like HTTP, UDP, TCP, ICMP etc. For example, one sensor can be dedicated for HTTP alone; another sensor can analyze UDP and ICMP and yet another one can analyze TCP traffic other than HTTP. This division can be done easily by configuring the libpcap filters on each of the sensors appropriately. The approach depends on the ability to divide the network traffic among the different sensors in a meaningful way, which can be another challenge. This approach would not work well when the attack spans over different protocols and would require a centralized analyzer to solve the problem.

The load balancing can also be based on the destination address. Different sensors would be processing the subset of the traffic bound to a

particular subnet and probably reporting to a central server. This is also called the distributed approach. Note that an IDS watching a subnet's traffic will be overloaded if the traffic to that subnet constitutes the most of the traffic. (say 90%).

2. A Partitioning approach

This is a direct solution from the first bottleneck. Since Network speeds are increasing faster than Processor speed, the Centralized solution has reached its limits [Ref]. Kruegel and Giovanni [Ref] come up with a slicing or partitioning approach for doing Intrusion detection on High-speed networks. As contrast to Load balancing, the traffic is partitioned meaningfully to a distributed set of sensors each assigned to a set of detection rules. The division of traffic has to be so that it guarantees detection of all attack scenarios. This can also be looked upon as a "divide and conquer" approach.

3. Sampling

When the IDS cannot process all the traffic on a high-speed network, it could do statistical sampling of the traffic and process that sample. This is not a dependable or attractive approach. Sampling is as bad as dropping. Although the attacker cannot guess whether his packet will be processed by the IDS or not, there is a non-negligible probability that the attacker can sneak in his attack without getting noticed. So this solution is not acceptable.

4. Fixed snap-len approach:

In this approach the IDS captures only 'N' bytes of traffic off the wire and analyzes that. The motivation for this approach is that a lot of attacks can be detected just by analyzing the IP, TCP/UDP and higher layer protocol headers which can be captured within, say, the first 100 bytes or so. As can be assumed this will disable the IDS from attacks which need the bytes greater than 100.

5. Adaptive Intrusion detection.

This approach takes into account the various analysis tasks that an IDS does and fixes priorities or values to them. Also benchmarking is done and the cost of each analysis task is found. The main idea of this approach then, is to do the most important tasks in the available 'on the fly processing time' (or inter packet arrival time) mentioned in section 2. In other words, this approach tries to maximize the value of the IDS at any operating conditions. This needs some performance monitoring by the sensor itself. It should keep statistics like inter-packet-arrival time, number of packets received/sec and number of packets dropped/sec. Libpcap [Ref] the packet capture library provides APIs to get number of packets dropped.

The philosophy of this approach is that more important analysis tasks should not be at the mercy of less important tasks and should be given more priority.

6. ASICs / FPGAs

ASICs (or Application Specific Integrated Circuits) are used a lot nowadays to create switches and routers. Research and work are in progress to make ASIC based IDS too. That would be a very big step towards high-speed intrusion detection. Research is also being done in order to use FPGA (Field Programmable Gate Arrays) to assist IDS. David and Didier [Ref] talks about performing Tcp-stream reassembly and state tracking using FPGA.

Another interesting work is by Franklin, Carver and Hutchings [Ref] in which they do FPGA based regular expression matching on bit streams. They based their regular expression on the snort database. A JHDL based compiler extracts regular expressions from snort database and make circuits that do the matching. FPGA acts as a co-processor to offload pattern matching work from the main processor doing the intrusion detection, thereby pushing the technology towards higher and higher bandwidths.

7. Network processors.

Network processors are also new in the field of NIDS. Intel's IXP 1200 network processors are an example. The design goals of this Network processor [11] included Intrusion detection on high-speed networks. The architecture of this network processor shows that it is well suited for highly parallel computations. It is equipped with six multithreaded, micro engines and a choice of Intel® Strong-Arm* 166, 200, or 232 MHz processor core. One way to approach (based on SNORT architecture) is to divide the different modules of the IDS on different micro engines so as to do the processing of packets in a pipelined fashion. For example, the modules can be packet capture, IP fragmentation reassembly, TCP stream reassembly, other preprocessors and finally the detection engine. I am part of the research team at Georgia Tech where we are implementing this approach.

8. Better String matching algorithms:

Depending on the algorithm used, the number of signatures or rules which involves pattern matching, it can become a bottleneck in IDS performance. To lessen the impact of this effect, advanced algorithms should be used [10].

Conclusion:

This document discussed the various challenges that the network intrusion detection systems faces with the ever increasing high speed networks. It also discussed the various solutions / approaches available. Not any of the solution is a panacea for the problem. Each approach has its own advantages/drawbacks associated and is to be selected appropriately.

References:

[1]: Rules of thumb in data engineering – Jim Gray, Prashant Shenoy. http://research.microsoft.com/~gray/papers/MS_TR_99_100_Rules_of_Thumb_in_Data_Engineering.pdf
[2]: S.McCanne, C. Leres and V.Jaconson. libpcap. ftp.ee.lbl.gov, 1994
[3]: Winpcap: http://winpcap.polito.it/contact.htm
[4]: V.Paxson. Bro: A system for detecting network intruders in real-time. Computer Networks 99.
[5]: Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. Usenix Lisa 99'. http://www.snort.org/docs/lisapaper.txt
[6]: Practical security issues with high-speed networks. John velissarios. Price water house coopers.
[7]: IDS Group Tests. http://www.nss.co.uk/ids/ids_edition_2.htm

[8]: G.Shipley and P.Mueller. Dragon claws its way to the top. Network computing.http://www.networkcomputing.com/1217/1217f2.html
[9]:Increasing performance in High speed NIDS.-Neil Desai. http://www.linuxsecurity.com/resource_files/intrusion_detection/Increasing_Performance_in_High_Speed_NIDS.pdf
[10]: Towards faster string matching for Intrusion detection – Coit, Staniford et. al. http://www.silicondefense.com/software/acbm/speed_of_snort_03_16_2001.pdf
[11]: http://developer.intel.com/design/network/products/npfamily/ixp1200.htm
[12]: Beyond Moore's law: Internet growth trends, Roberts, L.G. IEEE Computer, Vol.33, Iss.1, Jan 2000 Pages:117-119
[13] Stateful intrusion detection for high speed networks, Kruegel, Valeur, IEEE Symposium on Security and Privacy May 12 - 15, 2002 California, P 285.

Part 2:

Detect #1: Load-balancing-device Probe matches Snort's NMAP rule

Trace Log:
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-03:03:40.884488 64.152.70.68:80 -> 46.5.180.250:53
TCP TTL:49 TOS:0x0 ID:53592 IpLen:20 DgmLen:40
A* Seq: 0x200 Ack: 0x0 Win: 0x578 TcpLen: 20
[Xref => arachnids 28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-03:03:40.884488 64.152.70.68:53 -> 46.5.180.250:53
TCP TTL:49 TOS:0x0 ID:53593 IpLen:20 DgmLen:40
A* Seq: 0x201 Ack: 0x0 Win: 0x578 TcpLen: 20
[Xref => arachnids 28]

Source of the trace:
The trace file used was www.incidents.org/logs/Raw/2002.5.10. The file is in the binary format or tcpdump readable format generated by a Snort IDS with unknown ruleset.

Type of Event generator:
The above alert was generated when the trace file was processed by Snort IDS version 1.9.0, with the stable rule set downloaded on Nov 12, 2002. The default ruleset was used. All the preprocessors were enabled.
The variables EXTERNAL_NET and HOME_NET were set to 'any'.

Snort was run with the following options:
snort -r 2002.5.10 -c etc/snort.conf -l/LOGS/

The alert was generated by the following snort rule in scan.rules -

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP";flags:A;ack:0; reference:arachnids,28; classtype:attempted-recon;
sid:628; rev:1;)
```

A brief description of the alert format is given below:

SCAN nmap TCP:	This is the name of the alert.
Classification:	Attempted Information Leak; Priority: 2 This is the importance associated with this particular alert. The classification is done in the classification.config file and this particular alert is classified as attempted-recon, with priority-2
06/10-03:03:40.884488	Time at which the alert was generated and the packet was logged into the trace file.
64.152.70.68:80	Source IP Address: Source port
46.5.180.250:53	Destination IP Address: Destination port
TCP	Protocol
TTL: 49	Time to live. This IP header field is decremented at each hop and the usual (recommended by RFC 1700) value is 64. This implies the packet has made 15 hops when the IDS processed it.
TOS:0x0	Type of Service. 0 is default value.
ID: 53593	Identification number. This IP header value uniquely identifies the IP datagram.
IpLen: 20	IP Header length. The default and min value is 20.
DgmLen:40	Total length of the IP datagram.
A*	represents the FLAGS field of the TCP header. Only the ACK bit is set.
Seq: 0x201	TCP Sequence number
Ack: 0x0	TCP Acknowledgement number. It is unusual to have ACK# 0 for a non-SYN tcp segment. (Discussed later)
Win: 0x578	TCP Window size
TcpLen: 20	Length of TCP headers The IP datagram length is 40, IP Header length is 20, TCP Hdr length is 20. There is no TCP payload.
Xref => arachnids 28	This is a reference o the corresponding alert entry in Arachnids, an Intrusion Event database.

The hex dump of the concerned packets using tcpdump is shown below:

```
03:03:40.884488 64.152.70.68.http > 46.5.180.250.domain: . ack 0 win 1400
4500 0028 d158 0000 3106 54a2 4098 4644
2e05 b4fa 0050 0035 0000 0200 0000 0000
```

```
5010 0578 4402 0000 0000 0000 0000

03:03:40.884488 64.152.70.68.domain > 46.5.180.250.domain: . ack 0 win 1400
4500 0028 d159 0000 3106 54a1 4098 4644
2e05 b4fa 0035 0035 0000 0201 0000 0000
5010 0578 441c 0000 0000 0000 0000
```

Investigation of the trace:
Normally, the ACK number of a TCP segment is 0 only for a SYN segment. The subsequent segments will have the ACK number equal to the next expected sequence number. There can be a valid case when the ACK number can be zero, i.e. when the expected sequence number is in fact 0 due to wrapping of sequence number space, but this is a rare case. The relevant parts from the RFC 793 are given below:

```
<snip>
Acknowledgment Number: 32 bits

  If the ACK control bit is set this field contains the value of the next sequence
number the sender of the segment is expecting to receive. Once a connection is
established this is always sent.
  Please note in the following that all arithmetic on sequence numbers,
acknowledgment numbers, windows, et cetera, is modulo 2**32 the size of the
sequence number space. Also note that "=<" means less than or  equal to
(modulo 2**32).
<snip>
```

In such a case, the ISN (Initial sequence number) will be near 2**32-1 and when (2**32-1)-ISN bytes are transferred, there can be an ACK segment ack-ing with an acknowledgement number=0. This means that the next sequence number expected is 0. The probability is quite low for this case.
Besides, it can be noted that both the packets have consecutive sequence numbers. The sequence numbers of the packets can be noted to be consecutive. Also, the values are too small for a 32 bit sequence number field.

The IP header ID fields of both the packets are also consecutive. It seems that the application that crafted the packets just incremented the Seq num and the ID fields by one.
The above packets, most probably, are crafted and do not belong to an established TCP session. This could be confirmed by analyzing:

- whether the target replies with a RST segment.
- whether the 3 way handshake has already been established.

But the trace file does not contain enough information for confirming these. The trace file was generated by a Snort IDS with unknown ruleset. Only the packets that caused alerts were logged. Especially complete 3 way handshakes of TCP connections were not present. (This was confirmed using the command -

tcpdump -r 2002.5.10 -n 'tcp[13]&0x7!=0'. No complete 3 way handshakes were seen)
Such probes from the same source were seen repetitively over consecutive day's log files. (See Appendix 3). All of them had the ACK number = 0; therefore it looks very likely that these are crafted packets which do not belong to any established TCP sessions.

What can we conclude about the topology from the trace seen?
The TCP segments under consideration does not belong to an established session, as discussed above, and would not pass through a stateful firewall. Therefore, the firewall (if present) is a stateless one. Another possibility is that the IDS is placed outside the firewall.

Description of attack:
'SCAN nmap TCP' alert is generated by Snort when it receives a TCP segment with

- Only ACK bit set in the TCP Flags and
- The ACK number is 0.

According to information from Arachnids, this signature will only detect older versions of NMAP that set the TCP ACK number to zero and also that the intent of the packet is to check if a host is reachable. [1]

Such a scan can also be used to –

- Check the firewall configuration.
- Test whether the firewall is a stateful or stateless one.
- Check the unfiltered and filtered ports on the firewall.
- Check if the target is reachable (as mentioned above. ref: arachnids).
- Scan for DNS machines on the target network.

Only the ACK bit is set. So, if the firewall is stateless it would not be able to distinguish this packet with another one of an established TCP connection. Such firewalls make decision based on each individual packet without keeping any state information. The packets are passed or dropped based on source and destination ports (allowed / blocked services).

The source ports of the packets are 53 and 80, respectively. This is a usual method to bypass stateless firewalls. The NMAP tool from www.insecure.org can be used to generate such a packet.

Nmap -g allows the attacker to specify a source port number that should be used. Nmap -sA is used for performing an 'ACK scan'. Previous versions of NMAP uses ACK# 0 while performing this scan.
The relevant section from NMAP man page is given in the Appendix section.

Attack Mechanism:
An ACK scan using low source ports! That was my first conclusion on seeing this alert. But similar packets were quite frequent and had the same destination

IP address and port. There were no evident signs of scanning / reconnaissance activity. Moreover, a reverse name lookup on the source IP address provided proximitycheck2.allmusic.com. (Note the name - proximitycheck)

I searched for related information on the web and found out that certain load balancing boxes in fact does similar probing. Link Proof by Radware is such a device. These boxes do such probing for performing traffic redirection and load balancing. I contacted support engineer at Radware and got a document which gave more insight - appnote-proxdet.pdf.
According to the document:

> To measure, Radware will initiate the proximity detection probe by sending several packets (upto 4) to the destination network and learning the hops and latency based on the replies to the proximity detection probe. The proximity probes are a combination of IP, TCP and application layer probes (such as TCP ACKS and ICMP Echo Requests) to ensure accurate measurements. The reply will either be a response to an ICMP Echo Request or an error message generated by the remote network in response to the other proximity detection probe packets.

However, the trace does not contain the ICMP Echo Requests and UDP packets from the same source port and we can reason it because the IDS has not logged those packets.

There are no signs of any scanning using such probes. It can be noted that the destination IP address/port is constant, which is not the case with usual scans. Moreover, the reverse DNS shows the source to be music.com and the naming of the machines - proximitycheck1 and proximitycheck2 links the devices to be some sort of load balancing devices.(refer section : Probability that the address is spoofed). It may be considered as a false alarm since the signature is matching an event, which is not exactly the event of interest. The packets are in fact probes but with a different intent than the ACK scan (event of interest).
On the other hand, the firewall related information is obtained; i.e. whether the firewall is stateful or not, whether firewall is open to port 53 and port 80. Even if this probe was with non-malicious intent, a passive attacker (listening to the network traffic) can get such information.

Probability that the address is spoofed:
Probably not.
This is a probe packet, and the source is most probably expecting a reply. Therefore, the probability of the source IP address being spoofed is low.

Such probes from the same source IP were seen repetitively over consecutive day's log files and the TTL values were the same (TTL:49) (See Appendix).

A reverse lookup on 64.152.70.68 gave the following:
% nslookup 64.152.70.68
Non-authoritative answer:
68.70.152.64.in-addr.arpa name = proximitycheck2.allmusic.com.

A traceroute to 64.152.70.68 was done from http://www.above.net/

```
FROM www.above.net TO 64.152.70.68.
traceroute to 64.152.70.68 (64.152.70.68), 30 hops max, 40 byte packets
1 inside.fw1.sjc2.mfnx.net (208.184.213.129) 0.303 ms 0.247 ms 0.222 ms
2 99.ge2-0.er4b.sjc2.us.mfnx.net (64.124.216.11) 0.577 ms 0.367 ms 0.339 ms
3 so-4-2-2.mpr4.sjc2.us.mfnx.net (208.185.156.193) 0.526 ms 0.748 ms 0.544ms
4 pos6-0.mpr2.pao1.us.mfnx.net (208.185.175.162) 0.835 ms 0.819 ms 0.805ms
5 gigabitethernet6-0.edge1.paix-sjo1.Level3.net (209.245.146.157) 0.896 ms 0.843 ms
0.830 ms
6 GigabitEthernet3-1.core1.SanJose1.Level3.net (209.244.3.249) 1.194 ms 1.120 ms
1.139 ms
7 gige10-1.ipcolo3.SanJose1.Level3.net (64.159.2.105) 1.186 ms 1.198 ms 1.220 ms
8 proximitycheck2.allmusic.com (64.152.70.68) 5.558 ms 6.560 ms 4.535 ms
```

Similar alerts were found from the trace file for next day i.e. www.incidents.org/logs/Raw/2002.5.11 and are given below:

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/11-07:12:31.894488 63.211.17.228:80 -> 46.5.180.250:53
TCP TTL:49 TOS:0x0 ID:12785 IpLen:20 DgmLen:40
***A**** Seq: 0x17A Ack: 0x0 Win: 0x578 TcpLen: 20
[Xref => arachnids 28]
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/11-07:12:31.894488 63.211.17.228:53 -> 46.5.180.250:53
TCP TTL:49 TOS:0x0 ID:12786 IpLen:20 DgmLen:40
***A**** Seq: 0x17B Ack: 0x0 Win: 0x578 TcpLen: 20
[Xref => arachnids 28]
```

A traceroute to 63.211.17.228 from www.above.net gives:

```
FROM www.above.net TO 63.211.17.228.
traceroute to 63.211.17.228 (63.211.17.228), 30 hops max, 40 byte packets
1 inside.fw1.sjc2.mfnx.net (208.184.213.129) 0.294 ms 0.262 ms 0.216 ms
2 99.ge2-0.er4b.sjc2.us.mfnx.net (64.124.216.11) 0.350 ms 0.371 ms 0.333 ms
3 so-4-2-2.mpr4.sjc2.us.mfnx.net (208.185.156.193) 0.530 ms 0.511 ms 0.509ms
4 pos6-0.mpr2.pao1.us.mfnx.net (208.185.175.162) 2.393 ms 0.901 ms 0.801ms
5 gigabitethernet6-0.edge1.paix-sjo1.Level3.net (209.245.146.157) 0.894 ms 0.881 ms
3.495 ms
6 GigabitEthernet3-1.core1.SanJose1.Level3.net (209.244.3.249) 1.268 ms 1.104 ms
1.093 ms
7 ae0-56.mp2.SanJose1.Level3.net (64.159.2.161) 1.622 ms 1.656 ms 1.698 ms
8 so-0-1-0.mp2.Detroit1.Level3.net (64.159.0.198) 85.962 ms 85.998 ms86.04ms
9 gige9-1.hsipaccess1.Detroit1.Level3.net (64.159.0.210) 87.742 ms 86.056 ms87.07ms
10 proximitycheck1.allmusic.com (63.211.17.228) 88.698 ms 88.443 ms 90.33ms
```

Similar traffic from the same organization allmusic.com seems to have triggered both the alerts.

The nslookup/traceroutes give the names of the sources for the above alerts- proximitycheck1 & proximitycheck2. Another point is that Link Proof by Radware is a proximity detection device.

The document (appnote-proxdet.pdf) from Radware, in fact, mentions:

> One way to minimize any client concern is to provide the Radware devices with a DNS names such as proximity-device.company.com or quality-of-service-device.company.com, or network-proximity-measuring-device.company.com. Should a client detect the proximity probe, they may do a reverse DNS lookup and then be informed of the nature and source of the probe.

All this suggests that the chances of the source address being spoofed is pretty low.

Correlations:
A similar discussion can be found at Sans site [3]
Chris Brenton also has analyzed such scans which was seen at incidents.org [2]

Information regarding 64.152.70.68 from DShield.org :
IP Address: 64.152.70.68
HostName: proximitycheck2.allmusic.com
DShield Profile:
Country: US
Contact E-mail: spamtool@level3.com
Total Records against IP: 5717
Number of targets: 1279
Date Range: 2002-12-06 to 2002-12-06
Ports Attacked (up to 10): Port Attacks
53 14
37852 36

Evidence of active targeting:
It is clear that the probes are directed towards this particular machine. There is no sign of any scanning. Both the packets (alerts) have destination port = 53. However, But that does not prove that the target runs DNS server because the source (load balancing device) does not expect a DNS server to run on the target machine. It just needs a RST packet (if there is a DNS server) or an ICMP destination unreachable, if there is no server listening on port 53.

Severity:
Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality -

The target IP address has been sanitized by incidents.org.

The packets that caused the alerts had destination port = 53. But that does not prove that the target runs DNS server because the source (load balancing device) does not expect a DNS server to run on the target machine. It just needs a RST packet (if there is a DNS server) or an ICMP destination unreachable, if there is no server listening on port 53. But by analyzing the 2002.5.10, these packets were seen:

```
01:57:49.484488 63.70.83.162.2329 > 46.5.180.250.http:
P 1539858796:1539858855(59) ack 3321212235 win 8760 (DF)
10:08:20.434488 208.62.40.112.3641 > 46.5.180.250.http:
P 2626429556:2626429615(59) ack 14123919 win 17520 (DF)
```

Also analyzing the 2002.5.11, these packets were seen:

```
11:15:26.564488 216.30.135.34.1095 > 46.5.180.250.domain:
P 3994613107:3994613135(28) ack 1030056136 win 32120
<nop,nop,timestamp 4493315 389937016> (DF)
11:20:38.264488 211.21.238.234.1026 > 46.5.180.250.domain:
P 1891915402:1891915430(28) ack 1349524697 win 32120
<nop,nop,timestamp 7390274 389968166> (DF)
14:10:36.244488 211.21.238.234.1027 > 46.5.180.250.domain:
P 4072701345:4072701373(28) ack 3525246507 win 32120
<nop,nop,timestamp 8410059 390988042> (DF)
14:18:46.814488 216.30.135.34.1110 > 46.5.180.250.domain:
P 2751539674:2751539702(28) ack 4034113541 win 32120
<nop,nop,timestamp 5593309 391037126> (DF)
```

The above trace shows packets destined to HTTP and DNS servers and seems to be part of an established TCP session. However, the trace does not have the traffic from the server back to the client. Therefore, with some uncertainty, we can assume that the target is running DNS and HTTP, and might be an important server. Therefore, criticality is given as 4.

Lethality:
The above alert is classified as false alarm and the alerts were found to be probes from a load balancing device. Therefore, lethality is low.
Lethality = 1
System Countermeasures:
There is not much information about how well the particular host is patched.
System Countermeasures = 4
Network Countermeasures:
The network topology is unknown. If we assume that IDS was placed inside the firewall, then the firewall is stateless and the ports 80 and 53 are open. (Because otherwise the above probe would not have been detected in the first place).
It would be good to have a stateful firewall.
Network Countermeasures = 1

Severity = (4 + 1) - (4 + 1) = 0. Overall, the severity is low.

Defensive Recommendations:
The TCP nmap scans can be effectively blocked using a stateful firewall. Since the particular packet does not belong to an established session it will be dropped at the firewall.
The probes from the load balancing device (Link Proof) consisted of ICMP Echo requests, TCP Ack probes and other UDP packets. The ICMP echo requests can be blocked at the perimeter easily using a packet filter. A stateful firewall will be useful in blocking TCP ACK probes and similar TCP packets that manipulate the TCP flags to fool the perimeter defense.

Multiple choice question:
TCP ACK Scans can be used to
a) to get information about firewall configuration
b) crash vulnerable machines
c) hijack TCP sessions
d) buffer overflow
Answer: a

Reference:
[1] http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids28&view=event
[2] Email from Chris Brenton,
http://www.incidents.org/archives/intrusions/msg08129.html
[3] http://www.sans.org/y2k/031401.htm

Appendix-1: Answers to questions from intrusions@incidents.org
- what type of OS uses a default TTL of 64?
Linux, FreeBSDs use a default ttl value of 64.

> What would you expect the TTL value to be if NMAP was run from
a Windows 2000/XP host?

Nmap run from windows uses different ttl values at different times.
Nmap older versions used default values (depending on OS) but
this provided some information leak about the scanner. This was fixed
and now it uses a random value per execution (between 37 and 64)

Appendix-2: Relevant section from NMAP man page:
<snip>
-sA ACK scan: This advanced method is usually used to map out firewall rule sets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets. This scan type sends an ACK packet (with random looking acknowledgement/ sequence numbers) to the ports specified. If a RST comes back, the ports are classified as "unfiltered". If

nothing comes back (or if an ICMP unreachable is returned), the port is classified as "filtered".

-g <portnumber>
Sets the source port number used in scans. Many naive firewall and packet filter installations make an exception in their rule set to allow DNS (53) or FTP-DATA (20) packets to come through and establish a connection. Obviously this completely subverts the security advantages of the firewall since intruders can just masquerade as FTP or DNS by modifying their source port. Obviously for a UDP scan you should try 53 first and TCP scans should try 20 before 53.
<snip>

Appendix-3: Similar packets over consecutive days:
Similar traces/probes from the same destination IP address across consecutive day's log files. Some snippets are shown below.

It can be noted that it occurs repeatedly over some irregular intervals.
According to the document appnote-proxdet.pdf from Link Proof -
Once the device knows the hops and latency between its network(s) and the clients network, the best three contant delivery paths will be recorded in the dynamic table. This data is stored in the dynamic table for an adminitratively defined period of time. The repetitive probes may be due to expiry of entries in such a dynamic table.

```
2002.5.10
03:03:40.884488 64.152.70.68.http > 46.5.180.250.domain: . ack 0 win 1400
03:03:40.884488 64.152.70.68.domain > 46.5.180.250.domain: . ack 0 win 1400
03:03:40.984488 63.211.17.228.http > 46.5.180.250.domain: . ack 0 win 1400
2002.5.11 :
07:12:31.894488 63.211.17.228.http > 46.5.180.250.domain: . ack 0 win 1400
07:12:31.894488 63.211.17.228.domain > 46.5.180.250.domain: . ack 0 win1400
07:12:32.004488 64.152.70.68.http > 46.5.180.250.domain: . ack 0 win 1400
2002.5.12:
10:52:24.684488 63.211.17.228.http > 46.5.180.250.domain: . ack 0 win 1400
10:52:24.694488 63.211.17.228.domain > 46.5.180.250.domain: . ack 0 win1400
10:52:24.914488 64.152.70.68.http > 46.5.180.250.domain: . ack 0 win 1400
<entries snipped due to space constraints>
```

Detect #2: Malformed IGMP packets

Trace Log:
[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/08-10:52:39.026507 207.166.206.31 -> 207.166.206.31
PROTO002 TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref => url www.cert.org/advisories/CA-1997-28.html]
[Xref => cve CVE-1999-0016]

[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/08-10:52:39.026507 207.166.206.26 -> 207.166.206.26
PROTO002 TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref => url www.cert.org/advisories/CA-1997-28.html]
[Xref => cve CVE-1999-0016]

[**] [1:527:3] BAD TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/08-10:52:39.026507 207.166.206.37 -> 207.166.206.37
PROTO002 TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
[Xref => url www.cert.org/advisories/CA-1997-28.html]
[Xref => cve CVE-1999-0016]

Source of the trace:
The trace file used was www.incidents.org/logs/Raw/2002.10.8
The file is in the binary format or tcpdump readable format generated by a Snort IDS with unknown rule set.

Type of Event generator:
The above alert was generated when the trace file was processed by Snort IDS version 1.9.0, with the stable rule set downloaded on Nov 12, 2002. The default ruleset was used. All the preprocessors were enabled.
The variables EXTERNAL_NET and HOME_NET were set to 'any'. Snort was run with the following options: snort -r 2002.10.8 -c etc/snort.conf -l/LOGS/
The alert was generated by the following snort rule in scan.rules -

alert ip any any -> any any (msg:"BAD TRAFFIC same SRC/DST";
sameip; reference:cve,CVE-1999-0016;
reference:url,www.cert.org/advisories/CA-1997-28.html; classtype:bad-unknown;
sid:527; rev:3;)

A brief description of the alert format is given below: (based on the first alert)

BAD TRAFFIC same SRC/DST	This is the name of the alert.
Classification:	Potentially Bad Traffic; Priority: 2 This is the importance associated with this particular alert. The classification is done in the classification.config file and this particular alert is classified as potentially bad traffic, with priority-2
11/08-10:52:39.026507	Time at which the alert was generated and the packet was logged into the trace file.
207.166.206.31	Source IP Address
207.166.206.31	Destination IP Address
PROTO002	IGMP Protocol

TTL: 47	Time to live This IP header field is decremented at each hop and the usual (recommended by RFC 1700) value is 64. This implies the packet has made 17 hops when the IDS processed it.
TOS:0x0	Type of Service. 0 is default value.
ID: 0	Identification number This IP header value uniquely identifies the IP datagram.
IpLen: 20	IP Header length. The default and min value is 20.
DgmLen:28	Total length of the IP datagram.
Xref => url www.cert.org/advisories/CA-1997-28.html Xref => cve CVE-1999-0016	This is a reference to the corresponding alert entry in cve.mitre.org, an Intrusion Event database.

Corresponding packet trace using tcpdump:

```
10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60:
207.166.206.31 > 207.166.206.31: igmp query v2 [gaddr 240.0.1.168]
4500 001c 0000 0000 2f02 e2ec cfa6 ce1f
cfa6 ce1f 1164 fcf2 f000 01a8 0000 0000
0000 0000 0000 0000 0000 0000 0000

10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60:
207.166.206.26 > 207.166.206.26: igmp query v2 [gaddr 240.0.1.163]
4500 001c 0000 0000 2f02 e2f6 cfa6 ce1a
cfa6 ce1a 1164 fcf7 f000 01a3 0000 0000
0000 0000 0000 0000 0000 0000 0000

10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60:
207.166.206.37 > 207.166.206.37: igmp query v2 [gaddr 240.0.1.174]
4500 001c 0000 0000 2f02 e2e0 cfa6 ce25
cfa6 ce25 1164 fcec f000 01ae 0000 0000
0000 0000 0000 0000 0000 0000 0000
```

Tethereal dump:

```
Frame 1 (60 bytes on wire, 60 bytes captured)
   Arrival Time: Nov  8, 2002 10:52:39.026507000
   Time delta from previous packet: 0.000000000 seconds
   Time relative to first packet: 0.000000000 seconds
   Frame Number: 1
   Packet Length: 60 bytes
   Capture Length: 60 bytes
Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
   Destination: 00:00:0c:04:b2:33 (00:00:0c:04:b2:33)
   Source: 00:03:e3:d9:26:c0 (00:03:e3:d9:26:c0)
```

```
   Type: IP (0x0800)
   Trailer: 00000000000000000000000000000000...
Internet Protocol, Src Addr: 207.166.206.31 (207.166.206.31),
Dst Addr: 207.166.206.31 (207.166.206.31)
   Version: 4
   Header length: 20 bytes
   Total Length: 28
   Identification: 0x0000
   Flags: 0x00
      .0.. = Don't fragment: Not set
      ..0. = More fragments: Not set
   Fragment offset: 0
   Time to live: 47
   Protocol: IGMP (0x02)
   Header checksum: 0xe2ec (incorrect, should be 0x5054)
   Source: 207.166.206.31 (207.166.206.31)
   Destination: 207.166.206.31 (207.166.206.31)
Internet Group Management Protocol
   IGMP Version: 2
   Type: Membership Query (0x11)
   Max Response Time: 10.0 sec (0x64)
   Header checksum: 0xfcf2 (correct)
   Multicast Address: 240.0.1.168 (240.0.1.168)
```

Probability that the address is spoofed:
Very high. It can be noted that the source and destination IP addresses are the same. This suggests that the source IP address has a very high chance of being a spoofed one. Such a packet should never be seen on the wire.

Description of attack:
This can be classified as a DoS attempt using a malformed packet against hosts which process IGMP. It is a network wide attempt. The tcpdump of the entire set is given in the Appendix-1.

The packet is a malformed IGMP Version: 2 Membership Query. The query is for an invalid group address and if this is not correctly handled, it may cause problems. However, there are no known vulnerabilities related to such a malformed IGMP query. The tool used to craft this packet is not known.

Moreover, the attack packets have same source and destination IP addresses which is ***similar*** to the LAND attack signature. Land attack consists of a TCP Syn packet with same source and destination IP address/port, and has been known for a while and most of the systems do not have the LAND vulnerability.
But there is no information with regards to this specific case; i.e. source and destination IP are equal for an IGMP protocol. The attacker might be trying out new ways to do Denial of service on remote machines.

As mentioned in a HACK FAQ at nmrc.org [1]

5.6 How can I discover new DoS attacks?
New DoS attacks are fairly easy to discover. Flooding any service or system with malformed or excessive packets and observing the behavior will tell you if you've discovered something interesting.

Attack Mechanism:
The packets are malformed IGMP query packets. It is an IGMP group specific query for an invalid multicast group (240.0.1.168). The relevant part from the Tethereal dump is shown below:
Internet Group Management Protocol
IGMP Version: 2
Type: Membership Query (0x11)
Max Response Time: 10.0 sec (0x64)
Header checksum: 0xfcf2 (correct)
Multicast Address: 240.0.1.168 (240.0.1.168)

It is a stimulus packet. Here, the attacker has used "unicast transmission" to carry IGMP payload, which is unusual. The source IP is spoofed also; and is same as the destination IP address. It will appear to the end host that the packet originated from itself. The attacker does'nt expect any replies from the target; so spoofing and hiding his identity (ip address) is logical. (This can be related to the LAND attack, which is a TCP Syn packet with source IP address equals destination IP address; and source port equals destination port.) Besides, serving as a hiding method, having source IP equal to destination IP makes it appear to the end host that the packet originated from itself. Since such packets are not expected on the wire, there might be inconsistencies with hosts/routers in such cases (for the specific case with IGMP protocol?). Usually, the destination IP address for a IGMP group specific query is same as the group that is being queried. However, IGMP v3 has mentioned that hosts should be lenient to process even unicast packets reaching its interface.

A host must keep a table of all the groups that atleast one process belongs to, and a reference count of the processes belonging to the group. When a group specific query comes from a 'Querier' router, it sends a REPORT if it belongs to the group. (after checking its internal table)
The RFC suggests that the host do some validation of the queries before processing. RFC 2236 comments -
"query received" occurs when the host receives either a valid General Membership Query message, or a valid Group-Specific Membership Query message. To be valid, the Query message must be at least 8 octets long, and have a correct IGMP checksum. The group address in the IGMP header must either be zero (a General Query) or a valid multicast group address (a Group-Specific Query).
If the above validations are not performed, it might cause problems while processing invalid requests, depending on implementation details of the IP/IGMP stack. However, there is no known vulnerabilities related to this.

Lets look into some more detail.
IGMP is used to manage multicast sessions and it is required to be implemented by all hosts wishing to receive IP multicasts as part of the IP stack.
From RFC 2236, IGMP messages can be of 3 types -
1.*Membership Query*
This can be General query or a Group specific query. A general query sets the group address field of IGMP message to 0; where as a group specific query sets it to the address of the specific group.
2.*Membership Report*
3.*Leave Group*

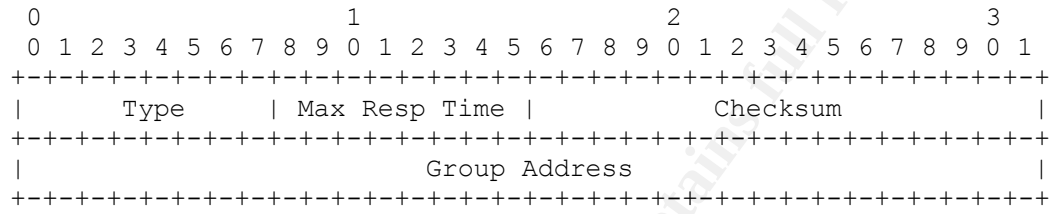


Fig.1 IGMP Message format

The multicast address present in the query (240.0.1.168) is not a valid multicast address. A valid multicast IP address lies in the class D range; i.e. 224.0.0.0 - 239.255.255.255

All the IGMP messages are send using multicasting, using the destination IP address for the appropriate multicast group. The destination MAC address of the message is also derived from the multicasting IP address.(shown below)

From RFC 2236:

Message Type	Destination Group
General Query	ALL-SYSTEMS (224.0.0.1)
Group-Specific Query	The group being queried

An example of a usual group specific query is as shown below. ***Note*** the similarity in the destination IP address and the group address. This packet was collected from a real network.

```
23:44:46.308414 128.61.136.2 > 239.255.255.250: igmp query v2
[max resp time 10] [gaddr 239.255.255.250] [tos 0xc0]  [ttl 1]
                        46c0 0020 0000 0000 0102 2bde 803d 8802
                        efff fffa 9404 0000 110a fefa efff fffa
                        0000 0000 0000 0000 0000 0000 0000
```

Another characteristic of a multicast packet is that the destination MAC address is derived from the destination IP address as mentioned in RFC 1054:

An IP host group address is mapped to an Ethernet multicast address by placing the low-order 23-bits of the IP address into the low-order 23 bits of the Ethernet multicast address 01-00-5E-00-00-00 (hex).

A valid ethernet address for a multicast packet should start with 01-00-5E. But, if you look at the tcpdump in Appendix 1, all the malformed packets have destination MAC 0:0:c:4:b2:33 which is incorrect. But, this is consistent with the statement that the attacker is using unicast transmission for sending the malformed IGMP to the target. The TTL value, discussed later, also supports this.
Looking in general at the raw file, we see that all the packets have destination MAC 0:3:e3:d9:26:c0 or 0:0:c:4:b2:33. This suggests that these are packets collected from an Ethernet network between two routers / packet filters.
This was pointed out by another student for his GCIA submissions.
(Post by Freeland Chew on incidents mailing list on 12/8/2002)

Let's take a look at the TTL value of the malformed IGMP packets.

RFC 2236 describes IGMP and states -
All IGMP messages described in this document are sent with IP TTL 1

TTL values is generally 1 for IGMP queries as well as other messages. This is because these messages are meant to be inside that LAN.
- Besides using unicast ip addresses, the attacker has used a higher value to make sure the packet reaches the target.
- Also, note that the IP identifier field is also 0 for all the packets. There is enough data to conclude that the packets are crafted.

Correlations:
There were no CVE or Bugtraq references found.
Another student, Daniel Wesseman, has done analysis on the same/similar alerts from incidents.org.

The attack bears similarity to the LAND attack as mentioned by the Snort alert. The CVE reference for LAND attack is CVE-1999-0016. [2]
Malformed IGMP packets causing denial of service has occured previously and can be found in the CVE database. CVE-1999-0918 - refers to DoS on Windows systems using malformed fragmented IGMP packets.
CVE-2001-0796 - refers to DoS using malformed IGMP packets (with small response delay/time). This affected SGI IRIX 6.5 and FreeBSD 3.0

Previous students have also explained malformed IGMP attacks.
Brent Deterding has analyzed fragmented IGMP or igmpnuke.
Buddy Smith has also analyzed fragmented IGMP attack.

Evidence of active targeting:
Low.
As it can be seen from the tcpdump output (Appendix 1), it is a network wide attempt. From the output it seems the malformed packets are sent to a random subset of machines in each subnet. Also, considering the IP addresses are sanitized, it is difficult to make any conclusions from looking at those IP addresses.

Severity:
Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality: 2
No information about the target is available. The attack spans a large portion of the network. There is a chance that it hits valid hosts; servers or workstations.

Lethality: 1
There is no information regarding any similar vulnerability. Queries with invalid group addresses do not cause any known problems. The Snort alert points to LAND attack, but that vulnerability was based on a TCP Syn packet with same source and destination IP addresses as well as ports. Vulnerabilities due to IP addresses alone being the same is not known, but is surely invalid.

System Countermeasures: 3
Hosts those do not support multicasting will not process such a packet; and for hosts that process such a packet, if they perform the checking suggested by the RFC, it is fine. RFC 2236 says -

> "query received" occurs when the host receives either a valid General Membership Query message, or a valid Group-Specific Membership Query message. To be valid, the Query message must be at least 8 octets long, and have a correct IGMP checksum. The group address in the IGMP header must either be zero (a General Query) or a valid multicast group address (a Group-Specific Query).

However, there is no available information about whether any of the target systems supports multicasting; or if they do the valid checking of the IGMP queries received.

Network Countermeasures: 1
The firewall or the router did not block these malformed packets. However, there is a Snort IDS running which logged alarms.

Severity = (2+1)-(3+1) = -1 (LOW)

Defensive Recommendations:
Defenses can be implemented at different levels :

- The Hosts supporting multicasting should perform the validity checks suggested by the RFC to make sure the query is valid. IGMP v3 also recommends using IPSEC in Authentication Header mode to protect against remote attacks by ensuring that IGMPv3 messages came from a system on the LAN (or, more specifically, a system with the proper key).
- In addition the host on receiving a receiving an IGMP packet, should check the MAC address. If it is not a multicast Ethernet address, i.e. with the prefix 01:00:5E, the host must drop the packet.
- Firewall can also be used to block/restrict IGMP messages, depending on the need for your network applications.

Multiple choice question:
IGMP protocol is an integral part of which protocol -
a. TCP
b. UDP
c. IP
d. TFTP
Answer :c

Reference:
[1] http://www.nmrc.org/faqs/hackfaq/hackfaq-5.html
[2] www.cert.org/advisories/CA-1997-28.html

Appendix - 1

TCPDump of mal-formed IGMP packets.
It can be seen that it is a network wide attack. Snippets from logs of adjacent days is shown below -

```
 /usr/sbin/tcpdump -n 'igmp' -e -r 2002.10.8
10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.206.31 > 207.166.206.31:
igmp query v2 [gaddr 240.0.1.168]
10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.206.26 > 207.166.206.26:
igmp query v2 [gaddr 240.0.1.163]
10:52:39.026507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.206.37 > 207.166.206.37:
igmp query v2 [gaddr 240.0.1.174]
10:52:39.036507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.207.45 > 207.166.207.45:
igmp query v2 [gaddr 240.0.1.184]
<snip>

marks% /usr/sbin/tcpdump -n 'igmp' -e -r 2002.10.9
02:55:34.036507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.199.38 > 207.166.199.38:
igmp query v2 [gaddr 240.0.0.231]
02:55:34.036507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.199.25 > 207.166.199.25:
igmp query v2 [gaddr 240.0.0.218]
02:55:34.036507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.199.32 > 207.166.199.32:
igmp query v2 [gaddr 240.0.0.225]
```

```
02:55:34.036507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.200.47 > 207.166.200.47:
igmp query v2 [gaddr 240.0.0.242]
<snip>
 marks% /usr/sbin/tcpdump -n 'igmp' -e -r 2002.10.11
19:02:51.796507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.71.211 > 207.166.71.211:
igmp query v2 [gaddr 240.0.3.94]
19:02:51.796507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.71.199 > 207.166.71.199:
igmp query v2 [gaddr 240.0.3.82]
19:02:51.796507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.71.192 > 207.166.71.192:
igmp query v2 [gaddr 240.0.3.75]
<snip>
marks% /usr/sbin/tcpdump -n 'igmp' -e -r 2002.10.13
02:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.38.167 > 207.166.38.167:
igmp query v2 [gaddr 240.0.3.146]
02:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.38.172 > 207.166.38.172:
igmp query v2 [gaddr 240.0.3.151]
02:22:18.726507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 ip 60: 207.166.38.177 > 207.166.38.177:
igmp query v2 [gaddr 240.0.3.156]
<snip>
```

Detect #3: Slapper worm

Trace Log:

```
Frame 1:
12:55:45.152257 00:05:00:d6:b8:80 00:08:20:db:1d:bc 0800 83: IP
(tos 0x0, ttl 45, length: 69) 210.70.60.99.1812 > MY.NET.0.0.1812:
[udp sum ok]  [|radius] (DF)
                4500 0045 0000 4000 2d11 bc2f d246 3c63
                xxyy 0000 0714 0714 0031 6eb0 0000 0000
                0b53 0000 cc18 2522 2600 0000 d1d2 d020
                1d00 0000 55af 958a 0105 0000 0000 0000
                217a 0200 00
Frame 2:
11:13:44.536955 00:05:00:d6:b8:80 00:08:20:db:1d:bc 0800 70: IP
(tos 0x0, ttl 45, length: 56) 210.70.60.99.1812 > MY.NET.0.0.1812:
[udp sum ok]  [|radius] (DF)
                4500 0038 0000 4000 2d11 bc3c d246 3c63
                xxyy 0000 0714 0714 0024 8875 0000 0000
                8bff 0000 7cd0 5abf 7400 0000 0000 0000
                0000 0000 0000 0000
```

A more detailed Tethereal dump of the packets is given in the Appendix-1.

Source of the trace:
The trace was obtained from the campus network.

Type of Event generator:
TCPDUMP was used to capture/display the above packets.

Explanation for the above format (based on Frame 1) :

12:55:45.152257	Time the packet was captured
00:05:00:d6:b8:80	Source MAC address
00:08:20:db:1d:bc	Destination MAC address
0800	Layer 3 protocol = IP
83	Frame length
tos 0x0	Type of service
ttl 45	Time to live
Length: 69	Length of the IP datagram
210.70.60.99	Source IP address
1812	Source port
MY.NET.0.0	Destination IP address
1812	destination port
UDP sum ok	Verification of UDP checksum
[|radius]	Radius - UDP port 1812 corresponds to the RADIUS protocol
(DF)	Don't fragment The DF flag of the IP header is set.

Some observances and points:

- TTL is 45. Seems like the source uses a default TTL value of 64 and the datagram has made 19 hops, when it was captured. Linux, FreeBSDs use a default ttl value of 64.
- IP Identification number is 0. RFC 791 says - "The identification field is used to distinguish the fragments of one datagram from those of another." The document seems to indicate ID value's usage primarily for Fragmentation only.

TCP IP Illustrated Vol1 - Richard Stevens quotes - The identification field uniquely identifies each datagram sent by a host.
The Article - http://www.sys-security.com/archive/bugtraq/ofirarkin2002-02.txt discusses this more and points out that Linux kernel uses ID = 0 in many cases, especially when DF is set. Later we will confirm that the source has Linux on it.

- Source and destination ports is 1812

RADIUS - Remote Authentication Dial-In User Service uses UDP protocol and port 1812. It is not unusual to have both source and destination ports 1812 but if we interpret the UDP payload, using the RADIUS packet format (Appendix - 3) we find out that Code = 0, Identifier = 0, Length = 0. This is unusual.

The minimum length is 20 and maximum length is 4096 for a RADIUS payload, and valid RADIUS codes are also given in appendix - 4. This points that this is an invalid packet. Later, we show that these packets belong to Slapper worm.

- Destination IP address - MY.NET.0.0

It can be noted that the host portion of the ip address is 0, which is unusual. However, RFC 1122 points out –

There is a class of hosts (4.2BSD UNIX and its derivatives, but not 4.3BSD) that use non-standard broadcast address forms, substituting 0 for -1. All hosts SHOULD recognize and accept any of these non-standard broadcast addresses as the destination address of an incoming datagram. A host MAY optionally have a configuration option to choose the 0 or the -1 form of broadcast address, for each physical interface, but this option SHOULD default to the standard (-1) form.

- There was an interesting case that happened, although it doesn't directly apply to the main analysis. TCPDUMP version that i was using (TCPDUMP version 3.6.3) crashed when it was used to analyze the above packets. The output looked like -

12:55:45.152257 210.70.60.99.radius > MY.NET.0.0.radius: rad-#0 41 [id 0] Attr[Term_action

It went to an infinite loop printing Term_action. The problem was related to a TCPDUMP bug where a RADIUS packet with zero length would cause an infinite loop. This happens only when you run TCPDUMP in display/interpreting mode. If TCPDUMP is used just in capture mode (- w), this problem is not present. The problem is corrected and is not present in the latest TCPDUMP version. The CVS log for TCPDUMP explains it. (Appendix)
This kind of a packet could very well be used as a DoS attack against any TCPDUMP listening on the network (in the display mode). A related attack is analyzed by Mark Cooper - GCIA 143.

Description of attack:
This is from a variant of the slapper worm - Slapper variant C2. The worm mainly affects Linux systems with Apache with mod_ssl installation having vulnerable OpenSSl libraries.
The worm apart from spreading itself to other vulnerable systems creates a peer to peer network of infected machines and makes it possible to do different types of Distributed-Denial-of-service attacks. Slapper worms exchange different messages; and the packets that is analyzed in this document belongs to that type. A brief description of Slapper C2 can be found at http://isc.incidents.org/analysis.html?id=175

Other versions differ slightly and the respective descriptions can be found at -
Slapper A: http://isc.incidents.org/analysis.html?id=167
Slapper B: http://isc.incidents.org/analysis.html?id=172
Slapper C: http://isc.incidents.org/analysis.html?id=173

The source code of the worm lists the systems that it targets -

{"Gentoo", "", 0x08086c34}, {"Debian", "1.3.26", 0x080863cc},
{"Red-Hat", "1.3.6", 0x080707ec},{"Red-Hat", "1.3.9", 0x0808ccc4},
{"Red-Hat", "1.3.12", 0x0808f614},{"Red-Hat", "1.3.12", 0x0809251c},
{"Red-Hat", "1.3.19", 0x0809af8c},{"Red-Hat", "1.3.20", 0x080994d4},
{"Red-Hat", "1.3.26", 0x08161c14},{"Red-Hat", "1.3.23", 0x0808528c},
{"Red-Hat", "1.3.22", 0x0808400c},{"SuSE", "1.3.12", 0x0809f54c},
{"SuSE", "1.3.17", 0x08099984},{"SuSE", "1.3.19", 0x08099ec8},
{"SuSE", "1.3.20", 0x08099da8},{"SuSE", "1.3.23", 0x08086168},
{"SuSE", "1.3.23", 0x080861c8},{"Mandrake", "1.3.14", 0x0809d6c4},
{"Mandrake", "1.3.19", 0x0809ea98},{"Mandrake", "1.3.20", 0x0809e97c},
{"Mandrake", "1.3.23", 0x08086580},{"Slackware", "1.3.26", 0x083d37fc},
{"Slackware", "1.3.26",0x080b2100}

Attack Mechanism:
The Slapper worm works in stages. The Infection stage involves scanning for vulnerable machines, exploiting and spreading the worm. Once the worm is passed and the target is infected, it spreads again and so on. Meanwhile, the infected machines create a peer to peer network of infected machines, exchanging information and messages.
The packets under consideration do not belong to the infection stage, but rather a worm on an infected machine is trying to exchange information to already compromised machines.

- Is it a stimulus or response? This is a stimulus packet.
- Affected Service: Slapper worm basically targets Web servers (Apache/Linux) with vulnerable OpenSSL libraries.
- Known Vulnerabilities/Exposures: All the machines in the target network are known to be well patched.
- Attack Intent: An instance of the worm is trying to send/exchange information to other infected machines or peers.

Let's analyze the packets a bit more.
Since the packets we are analyzing concerns the worm's communication part and not the infection part, the infection part is not discussed. Please find that at the links mentioned in the previous section.

Once the worm has infected a target, it exchanges information with other infected machines. There are different messages that can be exchanged. All the messages have the following format:
Each message has a low level header (llheader) and record(s)
- The llheader has type, checksum and id.
- Records - these can be different types of records like route records or list records etc
Each of these records start with a header
  - tag
  - id

- len
- seq

Different messages are identified by different tags as below -
known 'Tags' (from http://isc.incidents.org/analysis.html?id=167)

0x20: 'Info'	0x21: 'Open a bounce'	0x22: 'Close a bounce'
0x23: 'Send a message to bounce'	0x24: 'run a command'	0x25: not used
0x26: 'route'	0x27: not used	0x28: 'List'
0x29: 'UDP Flood'	0x2A: 'TCP Flood'	0x2B: 'IPv6 TCP Flood'
0x2C: 'DNS Flood'	0x2D: 'Email Scan'	0x41-0x47: 'Relay to Client'
0x70: 'Incoming Client'	0x71: 'Recieve the List'	0x72: 'Send the list'
0x73: 'Get my IP'	0x74: 'Transmit their IP'	

Trying to understand the worm payload of Frame - 1, according to the above headers structure -

0000 0000	0b53 0000	cc18 2522	2600 0000	
Type	Checksum	Id	Tag	
d1d2 d020	1d00 0000	55af 958a	01	05
id	Len	seq	sync	Hops

This message has a tag = 0x26, which means a route message. This probably contains some information about routing and has some metric 'hop with value 5.

Trying to understand the udp payload of Frame - 2, according to the above headers -

0000 0000	8bff 0000	7cd0 5abf	7400 0000
Type	Checksum	Id	Tag
0000 0000	0000 0000	0000 0000	
id	Len	seq	

This message has a tag = 0x74, and is sent when the worm on an infected machine does'nt know it's own IP.

Probability that the address is spoofed:

The packets being UDP packets and used for peer to peer messaging makes it a candidate for spoofing. But, apparently, the chances of spoofing are low.

1. The source IP address, 210.70.60.99, seems to be running vulnerable OpenSSL version (ref: http://www.openssl.org/news/secadv_20020730.txt) and also hosts Apache on Linux platform. So, the source is most probably an infected machine (and is not spoofing its address).

```
% telnet 210.70.60.99 80
Trying 210.70.60.99...
```

```
Connected to 210.70.60.99.
Escape character is '^]'.
HEAD / HTTP/1.1
Host: MY.NET.16.55

HTTP/1.1 200 OK
Date: Tue, 14 Jan 2003 05:47:34 GMT
Server: Apache/1.3.23 (Unix)  (Red-Hat/Linux) mod_python/2.7.6
Python/1.5.2 mod_ssl/2.8.7 OpenSSL/0.9.6b DAV/1.0.3 PHP/4.1.2
mod_perl/1.26 mod_throttle/3.1.2
Last-Modified: Thu, 26 Dec 2002 07:50:20 GMT
ETag: "11b8fd-4a31-3e0ab4bc"
Accept-Ranges: bytes
Content-Length: 18993
Connection: close
Content-Type: text/html
```

2. Both the packets have the same IP address and same TTL value also. This strengthens the fact that the IP is not spoofed.
3. A traceroute to 210.70.60.99 is given below. Note the number of hops,19, which is consistent with the TTL value of the packets (49), assuming a default of 64 as used by Linux.

```
% /usr/sbin/traceroute 210.70.60.99
traceroute to 210.70.60.99 (210.70.60.99), 30 hops max, 38 byte packets
 1  cc-cisco1-MY.NET (MY.NET.XX.1)  0.265 ms  0.215 ms  0.168 ms
 2  gateway2-MY.NET (MY.NET.YY.1)  0.216 ms  0.188 ms  0.191 ms
 3  sox-gw2-rtr.sox.MY.NET1 (MY.NET1.194.5)  0.657 ms  0.720 ms  0.582 ms
 4  atla.abilene.sox.net (199.77.193.10)  0.579 ms  0.632 ms  0.550 ms
 5  iplsng-atla.abilene.ucaid.edu (198.32.8.79)  10.598 ms  10.588 ms  10.593 ms
 6  kscyng-iplsng.abilene.ucaid.edu (198.32.8.81)  19.871 ms  33.154 ms  19.928 ms
 7  dnvr-kscy.abilene.ucaid.edu (198.32.8.13)  30.426 ms  30.448 ms  30.741 ms
 8  dnvrng-dnvr.abilene.ucaid.edu (198.32.11.110)  30.484 ms  39.435 ms  40.276 ms
 9  sttlng-dnvrng.abilene.ucaid.edu (198.32.8.49)  58.951 ms  58.984 ms  58.877 ms
10  TANET2-PWAVE.pnw-gigapop.net (198.32.170.42)  59.368 ms  59.024 ms 59.07ms
11  210.200.35.10 (210.200.35.10)  235.229 ms  235.662 ms  234.887 ms
12  tanet2-tanet.tanet2.net.tw (210.200.33.2)  235.836 ms  235.563 ms  236.455 ms
13  203.72.43.205 (203.72.43.205)  235.819 ms  235.759 ms  236.068 ms
14  140.111.230.253 (140.111.230.253)  240.556 ms  237.297 ms  236.963 ms
15  140.111.255.6 (140.111.255.6)  241.672 ms  239.421 ms  259.221 ms
16  210.240.0.254 (210.240.0.254)  239.611 ms  246.607 ms  240.983 ms
17  210.240.0.250 (210.240.0.250)  244.881 ms  357.911 ms  241.373 ms
18  210.240.0.26 (210.240.0.26)  272.160 ms  264.247 ms  252.010 ms
19  pc99.klcivs.kl.edu.tw (210.70.60.99)  358.892 ms  259.439 ms  254.195 ms
```

4. The source code of the worm also doesn't contain any indications of spoofing being done.

Correlations:
1. A query to dshield.org gave -

```
IP Address: 210.70.60.99
HostName: pc99.klcivs.kl.edu.tw
DShield Profile:
Country:
Contact E-mail:
Total Records against IP:  4
Number of targets:  4
Date Range: 2003-01-14 to 2003-01-14
Ports Attacked (up to 10):
Port Attacks
1812    1
80      2

Fightback:  not sent
Whois: inetnum:      210.70.0.0 - 210.71.127.255
netname:        TANET
country:        TW
descr:          Taiwan Academic Network
admin_c:        CY1-TW
tech_c:         ZL1-TW
mnt_by:         MAINT-TWNIC-NS
changed:        snw@www.edu.tw 980908
status:         ALLOCATED PORTABLE
source:         APNIC
start:          3527802880
end:            3527901183
diff:           98303
person:         Zi-Di Liu
address:        Taiwan Network Information Center
                Computer Center, Ministry of Education
                12th Fl, No. 106 Section 2, Hoping EastRd.
                Taipei
                TW
```

2. A sample detect of a Slapper version C worm's communication packet was given at incidents.org [1] (Appendix 3). Note the similarity in the UDP payload. That is also one containing the 'Route' Tag.

3. Another student, Edward W Ray, has submitted an analysis on Slapper worm detect. His analysis focuses on the scanning and the infection part.

Evidence of active targeting:
Low.

In this case, the worm is trying to communicate with other infected hosts in a random fashion. No signs of active targeting.

Severity:
Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality: 2
The worm is trying to communicate with infected machines in the target network at random.
Lethality: 4
The worm can potentially execute commands on the remote machine, if it can connect with an already infected machine; and even coordinate for Distributed DoS attacks.
System Countermeasures: 4
Since the campus network might have machines running vulnerable openssl with apache/linux. I verified that the main webservers are well patched.
Network Countermeasures: 1
No firewall; IDS present.
Severity = (2+4)-(4+1) = 1 (Low)

Defensive Recommendations:

1. Keep web servers patched. (OpenSSL/Apache/Linux)
2. Block UDP port 1812 (RADIUS) outgoing and incoming
   Usually internal machines do not use external RADIUS servers and vice versa.
3. Routers/Firewalls can drop directed broadcasts.
4. TCPDUMP versions should be patched, if there is a need to use.
5. Keep IDS running and signatures updated.

Snort rules had a rule for detecting Slapper worm communication as below -

```
alert udp $EXTERNAL_NET 2002 -> $HTTP_SERVERS 2002 (msg:"MISC
slapper worm admin traffic"; content:"|0000 4500 0045 0000 4000|";
offset:0; depth:10; classtype:trojan-activity;
reference:url,www.cert.org/advisories/CA-2002-27.html;
reference:url,isc.incidents.org/analysis.html?id=167; sid:1889; rev:3;)
```

content:"|0000 4500 0045 0000 4000|" seems to be trying to match the initial part of the IP header.

For example, 4500 0045 0000 4000 is common in all the below packets taken from the campus network here -

15:50:16.522194 IP 64.152.195.131.1812 > MY.NET.0.0.1812: [|radius] (DF)
4500 0045 0000 4000 2911 cabd 4098 c383
xxyy 0000 0714 0714 0031 3195 0000 0000
b2b8 0000 40fa f7e9 2600 0000 45c8 a22c

1d00 0000 3e10 f1c3 0105 0000 0000 0000
ef78 0200 00

Also, note the detect posted at http://isc.incidents.org/analysis.html?id=167 (Appendix 2); having the 4500 0045 0000 4000. Note that Snort rule has a small error in that a 0000 is extra. It should be corrected to content:"|4500 0045 0000 4000|". This rule also would be missing the worm packets which have size other than 69 bytes. I have notified the snort-sigs mailing list for the change needed.

Multiple choice question:
Slapper worm infects another vulnerable machine through -
a. UDP port 1812
b. TCP port 1812
c. TCP port 443
d. TCP port 80
Answer :c

Reference:
[1] http://isc.incidents.org/analysis.html?id=167

Appendix -1: Answers to questions from intrusions@incidents.org
1.Any special significance of this fact in this case? You seem to be using it to support the fact that the source of the packet is Linux. Is there anything else there?
No. It is an observation, which is used to support the statement that the source m/c is Linux.

2.So why is worm sending the packet there, if there are no compromised machines?
>* Attack Intent: An instance of the worm is trying to send/exchange
> information to other infected machines or peers.
So, is it "patched" or "infected" in this case?
I correct myself here. Since the campus network might have machines running vulnerable openssl with apache/linux. I only checked if the main webservers are well patched.

However, there was no slapper worm traffic outbound and also no slapper worm traffic destined to specific machine (with unicast ip address); i.e. there was no signs of compromised internal machines sending traffic outbound. (after monitoring the traffic for long time)

The traffic I am analysing is destined to a broadcast address, MY.NET.0.0, which is unusual for the worm's behaviour - The worm keeps a list of IP address (corresponding to already compromised m/cs) and exchanges machines.
The trace would look like -

(from http://isc.incidents.org/analysis.html?id=167)

14:14:23.705193 IP ns.lingv.ro.2002 > xx.yy.116.27.2002
0x0000 4500 0045 0000 4000 2b11 b5e2 c1e6 2582 E..E..@.+.....%.
0x0010 yyxx 741b 07d2 07d2 0031 f1ce 0000 0000 yxt......1......
0x0020 91be 0000 2bf0 3863 2600 0000 1395 277b ....+.8c&.....'{
0x0030 1d00 0000 c1a1 8f5c 0105 0000 0000 0000 ....\........
0x0040 9e2d 0000 00

-Note the unicast destination address.

In the considered case, it seems
- the worm is trying to send a broadcast message to a target network
thereby reaching all the compromised m/cs, if any.
OR
- the worm somehow has a network entry (MY.NET.0.0) in the list it maintains.
According to rfc 1122, IP addresses are not permitted to have the
value 0 or -1 for any of the <Host-number>. Also, there is no host misconfigured
with this IP address.

Appendix – 2

```
Frame 1 (83 bytes on wire, 83 bytes captured)
    Arrival Time: Jan  8, 2003 12:55:45.152257000
    Time delta from previous packet: 0.000000000 seconds
    Time relative to first packet: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 83 bytes
    Capture Length: 83 bytes
Ethernet II, Src: 00:05:00:d6:b8:80, Dst: 00:08:20:db:1d:bc
    Destination: 00:08:20:db:1d:bc (00:08:20:db:1d:bc)
    Source: 00:05:00:d6:b8:80 (00:05:00:d6:b8:80)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 210.70.60.99 (210.70.60.99), Dst Addr: MY.NET.0.0
(MY.NET.0.0)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 69
    Identification: 0x0000
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 45
    Protocol: UDP (0x11)
    Header checksum: 0xbc2f (correct)
```

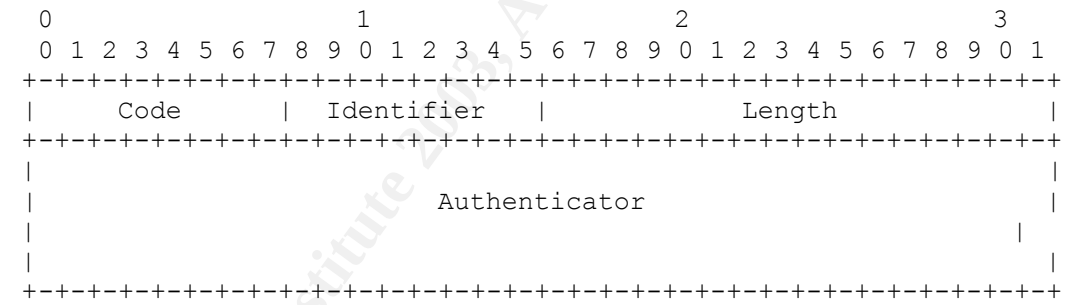
```
    Source: 210.70.60.99 (210.70.60.99)
    Destination: MY.NET.0.0 (MY.NET.0.0)
User Datagram Protocol, Src Port: 1812 (1812), Dst Port: 1812 (1812)
    Source port: 1812 (1812)
    Destination port: 1812 (1812)
    Length: 49
    Checksum: 0x6eb0 (correct)
Radius Protocol
    Code: Unknown (0)
    Packet identifier: 0x0 (0)
    Length: 0
    Authenticator

Frame 2 (70 bytes on wire, 70 bytes captured)
    Arrival Time: Jan  9, 2003 11:13:44.536955000
    Time delta from previous packet: 80279.384698000 seconds
    Time relative to first packet: 80279.384698000 seconds
    Frame Number: 2
    Packet Length: 70 bytes
    Capture Length: 70 bytes
Ethernet II, Src: 00:05:00:d6:b8:80, Dst: 00:08:20:db:1d:bc
    Destination: 00:08:20:db:1d:bc (00:08:20:db:1d:bc)
    Source: 00:05:00:d6:b8:80 (00:05:00:d6:b8:80)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 210.70.60.99 (210.70.60.99), Dst Addr: MY.NET.0.0
(MY.NET.0.0)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 56
    Identification: 0x0000
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 45
    Protocol: UDP (0x11)
    Header checksum: 0xbc3c (correct)
    Source: 210.70.60.99 (210.70.60.99)
    Destination: MY.NET.0.0 (MY.NET.0.0)
User Datagram Protocol, Src Port: 1812 (1812), Dst Port: 1812 (1812)
    Source port: 1812 (1812)
    Destination port: 1812 (1812)
    Length: 36
    Checksum: 0x8875 (correct)
Radius Protocol
    Code: Unknown (0)
    Packet identifier: 0x0 (0)
```

Length: 0 Authenticator

Appendix – 3

```
14:14:23.705193 IP ns.lingv.ro.2002 > xx.yy.116.27.2002: type: 0
chksum: be91 id: 6338f02b | tag: 26 id: 7b279513 len: 29 seq:
5c8fa1c1 | route sync=1 hops=5 server=0 links=11678 [slapper]
(DF)
0x0000 4500 0045 0000 4000 2b11 b5e2 c1e6 2582 E..E..@.+.....%.
0x0010 yyxx 741b 07d2 07d2 0031 f1ce 0000 0000 yxt......1......
0x0020 91be 0000 2bf0 3863 2600 0000 1395 277b ....+.8c&.....'{
0x0030 1d00 0000 c1a1 8f5c 0105 0000 0000 0000 .......\........
0x0040 9e2d 0000 00 .-...
14:14:23.733149 IP 217.167.206.74.2002 > xx.yy.116.27.2002: type:
0 chksum: c3 id: de43a4f1 | tag: 26 id: 613538b0 len: 29 seq:
4cf7e56f | route sync=1 hops=5 server=0 links=11692 [slapper]
(DF)
0x0000 4500 0045 0000 4000 2d11 f358 d9a7 ce4a E..E..@.-..X...J
0x0010 yyxx 741b 07d2 07d2 0031 6015 0000 0000 yxt......1`.....
0x0020 c300 0000 f1a4 43de 2600 0000 b038 3561 ......C.&....85a
0x0030 1d00 0000 6fe5 f74c 0105 0000 0000 0000 ....o..L........
0x0040 ac2d 0000 00
```

Appendix – 4: RADUIS Packet format



Appendix – 5: RADIUS Codes (decimal) are assigned as follows:

1	Access-Request	2	Access-Accept
3	Access-Reject	4	Accounting-Request
5	Accounting-Response	11	Access-Challenge
12	Status-Server (experimental)	13	Status-Client (experimental)
255	Reserved		

Appendix - 6

http://www.tcpdump.org/cvs-log/2002-01-21.10:16:48.html

File: tcpdump/print-radius.c
Revision: 1.7;
Date: 2001/06/18 09:16:28;

Author: guy;
Lines: (+25 -6)

Description: Don't use "sizeof()" to find the minimum RADIUS
packet length (although, as it's a multiple of 4, it's probably
not a problem on the most common offender here, GCC-on-ARM).
Hand to the code that dissects RADIUS attributes, as the length
of the attributes, min(payload length, captured payload length, length from
header) minus the size of the fixed-length fields in the RADIUS packet.
When printing RADIUS attributes, quit if we find one with a zero length,
rather than looping infinitely.

Analyze This!

1. Executive Summary:

A security audit was performed as requested by the GIAC Institute based on the alert, scan and OOS log files for the period Jan 16th – Jan 20th 2003. The files were downloaded from http://www.incidents.org/logs/ as per the requirement and are listed in the next section. Various issues were identified and defensive recommendations were made. Some important issues which require immediate actions are listed below. Detailed analysis on top critical alerts as well as suspicious internal hosts can be found in the later sections.

As a general rule, it is recommended that services on hosts, which are not needed be turned off.

- A list of machines is provided in section 11- 'Insight into certain insider machines'. These machines need to be checked for potential compromise.
- Installing a state-ful firewall is recommended. If that is not possible according to the university limitations, it is advised to have router access control lists or packet filters restricting/blocking traffic on certain ports. For example, TFTP is a non secure protocol and external machines should not be allowed to access internal TFTP servers.
- Egress filtering is also recommended. There are instances where internal hosts are suspected to have spoofed their IP addresses, which can be prevented by blocking outbound packets with external source IP address.
- Some machines are found to be misconfigured with improper IP addresses. They have used 192.0.0.0/24 address space instead of the private address space- i.e. 192.168.0.0/16.
- It is also recommended to update IDS configurations so as to have better detection.
    1. For example, spp_frag2 preprocessor should be used in place of spp_defrag in the snort. The later one is an obfuscated preprocessor with limitations.

2. Snort's configuration should understand the private IP addresses as part of HOME_NET so as to reduce the number of false alarms.

2. Files analyzed:

The files analyzed are given below. The files belong to the period from Jan 16th – Jan 20th 2003. Note that 6 OOS files had to be analyzed to cover the time period.

alert.030116	scans.030116	OOS_Report_2003_01_16_30391.txt
alert.030117	scans.030117	OOS_Report_2003_01_17_22332.txt
alert.030118	scans.030118	OOS_Report_2003_01_18_6261.txt
alert.030119	scans.030119	OOS_Report_2003_01_19_19130.txt
alert.030120	scans.030120	OOS_Report_2003_01_20_10420.txt
		OOS_Report_2003_01_21_8590.txt

3. Host Profile:

The alert and OOS files contain logs with home IP address sanitized.
But scan files have not been sanitized. Comparing the corresponding logs across the files, it can be noted that HOME NETWORK is 130.85/16.

This section explains the host/IDS/firewalls profile gathered from information available from the log files. A number of conclusions are made about the topology of the Home network, configurations of Firewalls, IDSs, hosts, and servers. This makes an important contribution towards the total analysis and helps eliminate many false alarms.

1. Firewall:
   Firewall is an important component of the network security framework of the organization. There is no information whether a firewall is in place. In case there is one, it is a stateless firewall.
   It can be noticed that 'NMAP TCP PING' alerts are generated. For example,

```
01/18-20:55:48.582409 [**] NMAP TCP ping! [**] 61.144.229.242:80 -> MY.NET.146.47:80
01/18-20:55:53.242589 [**] NMAP TCP ping! [**] 210.22.4.18:80 -> MY.NET.146.47:80
```

   The snort signature used for this alert might be:

```
alert tcp EXTERNAL_NET any -> MY.NET/16 any ( flags: A; ack: 0; msg:"NMAP TCP ping!";)
```

   The same or similar alert from the latest snort rule set is given below:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP";flags:A;ack:0;
reference:arachnids,28; classtype:attempted-recon; sid:628; rev:1;)
```

   The packets which trigger this alert, most probably, are not part of any established tcp sessions. These packets seem to be part of an ACK scan done using NMAP tool. Assuming the IDS that generated this alert is placed

inside the firewall/router; we can comment that the firewall is not a stateful firewall.

Another point that can be noticed from the alerts/logs that NAT (Network Address Translation) is being used. This is being done at the border router or a firewall.

```
01/16-13:10:34.073147  [**] TCP SRC and DST outside network
[**] 192.168.0.147:1652 -> 216.49.88.100:80
01/16-13:11:05.636049  [**] TCP SRC and DST outside network
[**] 192.168.0.147:1653 -> 216.49.88.100:80
01/16-13:11:20.457685  [**] TCP SRC and DST outside network
[**] 192.168.0.147:1655 -> 207.217.120.137:110
```

2. IDS.
   The log files analyzed are from Snort IDS, with a fairly standard (unknown) rule base. The snort configuration has portscan, http_decode (and Unicode analysis), fragmentation reassembly preprocessors turned on. The HOME_NET variable is probably set to MY.NET/16. (130.85/16)
   It can be noted that there are alerts private source/destination IP address i.e. 192.168/16 or 10/8. This points that the IDS is placed inside the firewall or router (possibly in the DMZ). However, the log files could have been from multiple IDSs too.

3. From the log files, it is also possible to make a list of different servers – Web server, mail server, DNS server, FTP server, which is useful for analysis purpose.
   The log files contain only alerts and not all packets seen on wire. Therefore, it cannot be said whether a machine actually runs a particular service from the alert in all cases. For e.g. From an alert, due to a SYN packet to a certain service on a machine we may not conclude that the machine does run that service; where as alerts which need an established connection points out that the destination machines run the particular service. Alerts like “IIS Unicode attack detected” need an established TCP session; hence there is a high chance that the target m/c runs web server. Similarly, alerts which have source port 80 could be from a machine running httpserver.

Type	Description
TFTP Servers	• Usually listen on UDP port 69. • Store router configuration files • No authentication and hence less secure MY.NET.111.219,MY.NET.111.231,MY.NET.111.230,MY.NET.111.235 MY.NET.111.232
Web Servers	• Usually listen on tcp port 80 MY.NET.150.16, MY.NET.84.193, MY.NET.70.231,MY.NET.29.3, MY.NET.179.77,MY.NET.157.52,MY.NET.70.207,MY.NET.168.140, MY.NET.157.52,MY.NET.130.40,MY.NET.137.18,MY.NET.132.42,

	MY.NET.136.2,MY.NET.150.228,MY.NET.99.36,MY.NET.157.12 (All the 600+ are not shown here)
FTP Servers	FTP uses port 21 (for the control messages) MY.NET.140.3
DNS Servers	DNS servers listen to TCP/UDP port 53. MY.NET.137.7
Mail Servers	SMTP uses port 25, IMAP uses 143 and POP uses 110. SMTP: MY.NET.6.40, MY.NET.145.9, MY.NET.139.230,MY.NET.140.236 POP: MY.NET.70.50, MY.NET.70.49
LPR servers	Line printer servers use tcp port 515. MY.NET.132.42

4. Alert listing

The alerts files were analyzed using methods, which is described later. The table below lists the different alerts that occurred during the period (Jan 16-20) and their frequency.

A brief analysis/discussion on the most frequent 15 alerts is given. The analysis addresses factors like chances that the alert is a false alarm, and whether spoofing is involved, and the severity or importance of the alert.

#	Alert	Frequency
1	High port 65535 tcp - possible Red Worm – traffic	67902
2	Watchlist 000220 IL-ISDNNET-990517	41781
3	Russia Dynamo - SANS Flash 28-jul-00	39207
4	SMB Name Wildcard	29382
5	TFTP - External UDP connection to internal tftp server	20698
6	spp_http_decode: IIS Unicode attack detected	17819
7	High port 65535 udp - possible Red Worm – traffic	4011
8	Incomplete Packet Fragments Discarded	2863
9	Possible trojan server activity	2688
10	spp_http_decode: CGI Null Byte attack detected	1900
11	EXPLOIT x86 NOOP	1757
12	IDS552/web-iis_IIS ISAPI Overflow ida nosize	1608
13	Queso fingerprint	1573
14	Watchlist 000222 NET-NCFC	1503
15	SUNRPC highport access!	1227
16	TCP SRC and DST outside network	906
17	Port 55850 tcp - Possible myserver activity - ref. 010313-1	389
18	Null scan!	365
19	External POP to HelpDesk MY.NET.70.49	349
20	IRC evil - running XDCC	334
21	TFTP - External TCP connection to internal tftp server	311
22	External POP to HelpDesk MY.NET.70.50	276
23	External RPC call	169
24	ICMP SRC and DST outside network	163

25	connect to 515 from inside	136
26	SMB C access	133
27	TFTP - Internal UDP connection to external tftp server	95
28	NMAP TCP ping!	87
29	EXPLOIT x86 setuid 0	62
30	EXPLOIT x86 setgid 0	50
31	Port 55850 udp - Possible myserver activity - ref. 010313-1	48
32	External FTP to HelpDesk MY.NET.70.49	24
33	External FTP to HelpDesk MY.NET.70.50	22
34	EXPLOIT x86 NOPS	21
35	Tiny Fragments – Possible Hostile Activity	18
36	connect to 515 from outside	14
37	PHF attempt	12
38	Attempted Sun RPC high port access	11
39	EXPLOIT NTPDX buffer overflow	9
40	DDOS shaft client to handler	7
41	TFTP - Internal TCP connection to external tftp server	6
42	HelpDesk MY.NET.83.197 to External FTP	6
43	External FTP to HelpDesk MY.NET.83.197	4
44	RFB - Possible WinVNC - 010708-1	3
45	Fragmentation Overflow Attack	2
46	EXPLOIT x86 stealth noop	2
47	SYN-FIN scan!	1
48	Probable NMAP fingerprint attempt	1
49	FTP DoS ftpd globbing	1
50	DOS Real Server template.html	1
51	Back Orifice	1

5. Detect analysis and correlations

Alert #1: High port 65535 tcp - possible Red Worm – traffic
Alert frequency – 67904 Category: Worm traffic

The Red worm or Adore worm exploits vulnerabilities in rpc.statd, bind, LPRng, and wuftpd26. It creates a backdoor on the infected machines on TCP port 65535. It is explained by Anthony Dell in “Adore Worm – Another Mutation” [1]. The worm replaces the ‘Kernel Log Daemon’, klogd with another version which implements a backdoor. This activates when it receives a ping with certain number of bytes (77 bytes).The worm opens a shell on port 65535 and will be used to transfer sensitive data like /etc/passwd or /etc/shadow
A snippet from “Adore Worm – Another Mutation” [1]:

```
The worm captures important system information, including userids and
running processes,  and sends the information to two different e-mail
addresses (either adore9000@21cn.com and adore9000@sina.com, or
```

```
adore9001@21cn.com and adore9001@sina.com). This worm also randomly
generates the first two octets of an IP address and then scans that
entire subnet for any other vulnerable systems. Once the worm finds a
vulnerable system, it infects the new system and the worm propagates
again.
```

Besides red worm, TCP port 65535 is also associated with trojans RC1 and ICE.

The signature used for this detect is not present in the latest rule set of Snort. The rule which triggered could have been as given below, described by Lorraine Weaver [2].

```
alert TCP any any -> any 65535 (msg:"High port 65535 tcp - possible Red
Worm – traffic";)
alert TCP any 65535 -> any any (msg:"High port 65535 tcp - possible Red
Worm – traffic";)
```

Almost 20 internal machines seem to be potentially affected by the Red Worm. The top 5 destinations (internal machines having backdoor port open) are given below. All the affected machines and especially the ones below are to be investigated further.

Top talkers (internal)

MY.NET.84.151	MY.NET.88.193	MY.NET.198.220	MY.NET.88.238	MY.NET.6.40

MY.NET.84.151

More than 130 IP addresses seem to be connecting to the backdoor on MY.NET.84.151. The top 5 external IP's connecting are:

IP Address	Rev name lookups
217.136.73.54	54.73-136-217.adsl.skynet.be
62.147.242.129	lns-p19-25-62-147-242-129.adsl.proxad.net
80.200.147.156	156.147-200-80.adsl.skynet.be
217.225.205.66	pD9E1CD42.dip.t-dialin.net
212.95.85.172	ip-85-172.evc.net

MY.NET.84.151 was not involved in other alerts.

MY.NET.88.193

33 different source IP's connect to the backdoor port (TCP 65535). The top 5 IP's involved and the reverse name lookups are:

IP Address	Rev name lookups
81.48.122.172	AClermont-Ferrand-201-1-4-172.abo.wanadoo.fr
80.15.81.167	AClermont-Ferrand-201-1-3-167.abo.wanadoo.fr
62.212.112.98	chtiot.net2.nerim.net
81.50.44.197	AClermont-Ferrand-201-1-5-197.abo.wanadoo.fr
80.11.124.214	AClermont-Ferrand-201-1-1-214.abo.wanadoo.fr

MY.NET.88.193 was not involved in other alerts.

MY.NET.198.220

The alerts show that two machines, 217.136.153.233 and 80.200.147.21 actively connecting (multiple connections) to tcp port 65535 on MY.NET.198.220.

IP address	Rev Name lookups
217.136.153.233	233.153-136-217.adsl.skynet.be
80.200.147.21	21.147-200-80.adsl.skynet.be

MY.NET.198.220 was involved in the following alerts:
- Actively scanning external hosts for port 6667. IRC server listens to this port.
- 112 alerts: IRC evil - running XDCC [**] MY.NET.198.220:4772->Ext_addr:6667
- 113 alerts:
  EXPLOIT x86 NOOP [**] 24.189.153.176:port -> MY.NET.198.220:1070

MY.NET.6.40
MY.NET.6.40 seems to be an smtp server. It was involved in the following other alerts:

- 578 alerts : Queso fingerprint (discussed later)
- 177 alerts: Watch list 000222 NET-NCFC
- 16 alerts: Null scan!
- 1 alert: NMAP TCP ping!
- Possible Trojan server activity
- Port 55850 tcp - Possible myserver activity - ref. 010313-1

Top talkers (external):

IP Address	Rev name lookups
217.136.73.54	54.73-136-217.adsl.skynet.be
62.147.242.129	lns-p19-25-62-147-242-129.adsl.proxad.net
80.200.147.156	156.147-200-80.adsl.skynet.be
217.225.205.66	pD9E1CD42.dip.t-dialin.net
212.95.85.172	ip-85-172.evc.net

Correlations:
A previous candidate, Lorraine Weaver has covered this alert in the GCIA practical [2].

Defensive recommendations:

- MY.NET.84.151, MY.NET.88.193 and MY.NET.70.176 need to investigate further for potential worm infection.
- The patches for BIND, rpc.statd, LPRng, and wu-ftp vulnerabilities that this worm exploits should be applied. They were released a long time back.

Reference:
[1] Adore Worm – Another Mutation - J. Anthony Dell
http://www.sans.org/rr/threats/mutation.php
[2] Lorraine Weaver. http://www.giac.org/practical/Lorraine_Weaver_GCIA.zip

Alert #2: Watchlist Alerts

Watch list 000220 IL-ISDNNET-990517	Alert-frequency : 41780

These alerts belong to watch list category. The purpose of these alerts is to detect traffic from IP addresses belonging to the watch list. Suspicious or blacklisted IP addresses are kept in the list.
Watch list 000220 IL-ISDNNET-990517 seems to be using 212.179/16 as the suspicious set of IP addresses. The Snort rule used could be -

```
alert ip 212.179/16 any -> $HOME_NET any (msg:"Watch list 000220 IL-ISDNNET 990517";)
```

```
Information from Dshield.org:
inetnum:      212.179.0.0 - 212.179.255.255
netname:      IL-ISDNNET-990517
descr:        PROVIDER
descr:        ISDNet LTD
country:      IL
```

The reverse name lookups for the top talkers (5) are shown below.

Alert #	IP Address	Rev name lookups
16197	212.179.1.145	fr-c27145.kbm.org.il
5889	212.179.56.252	bzq-179-56-252.cust.bezeqint.net
4098	212.179.107.228	bzq-179-107-228.dcenter.bezeqint.net
3619	212.179.105.69	cablep-179-105-69.cablep.bezeqint.net
3581	212.179.98.160	cablep-179-98-160.cablep.bezeqint.net

The top destination ports for the alert are shown below:

Alert #	Destination port
22345	1214
3790	2418
3606	4068
1964	3011
1425	1625

Watch list 000222 NET-NCFC	Alert-frequency - 1503

Watch list 000222 NET-NCFC seems to be using 159.226/16 as the suspicious set of IP addresses. 'NCFC' is the Net name for The Computer Network Center Chinese Academy of Sciences. The Snort rule used could be -

```
alert ip 159.226/16 any -> $HOME_NET any (msg: "Watchlist 000222 NET-NCFC ";)
```

A lookup for 159.226.0.0 on www.arin.net gives:

```
OrgName:    The Computer Network Center Chinese Academy of Sciences
OrgID:      CNCCAS
NetRange:   159.226.0.0 - 159.226.255.255
CIDR:       159.226.0.0/16
```

NetName: NCFC

Alert#	IP Address	Description
452	159.226.119.3	MY.NET.153.143 seems to be accessing 159.226.119.3's web service. The traffic is most probably non malicious.
358	159.226.99.14	MY.NET.153.126 seems to be accessing 159.226.99.14's web service. The traffic is most probably non malicious.
156	159.226.39.166	MY.NET.87.123 seems to be accessing 159.226.39.166's web service. The traffic is most probably non malicious.
107	159.226.236.23	MY.NET.88.250 and MY.NET.110.168 seems to be accessing 159.226.236.23's web service. The traffic is most probably non malicious.
76	159.226.120.14	159.226.120.14 is sending packets to smtp port on MY.NET.6.40, which is one of the mail servers. 159.226.120.14 has name - mail.nigpas.ac.cn and runs SMTP, IMAP4 and POP3. Most probably this is a mail server. Traffic is most probably non malicious.

Alert #3: Russia Dynamo - SANS Flash 28-jul-00
Alert-frequency – 39205 Category: Reconnaissance

This alert falls under the class of Watch list alerts. The signature was written to trigger on seeing any incoming/outgoing packets from/to a suspicious network – 194.87.6/24 (dol.ru). The alert was triggered 39205 times and the entire set of alerts was triggered from traffic between MY.NET.105.204: 4657 and 194.87.6.86: 2244. That is significant traffic and was exchanged over a period of 5 hours 35 minutes.
From the logs there is no information whether the packets are TCP or UDP.
The alert is most probably similar to:

```
alert IP $HOME_NET any -> 194.87.6.0/24 any (msg:"Russia Dynamo – SANS Flash 28-jul-00";)
alert IP194.87.6.0/24 any -> $HOME_NET any (msg:"Russia Dynamo – SANS Flash 28-jul-00";)
```

More investigation is recommended to verify whether MY.NET.105.204 is compromised.
Summary of other alerts involving MY.NET.105.204:

- Alert files show that 130.13.105.43 had tried IIS Unicode attack on MY.NET.105.204.
- MY.NET.105.204 has also triggered 3600+ 'Watch list 000220 IL-ISDNNET-990517' alerts. This alert was explained in the previous section.

194.87.6.86 was resolved to 86.6.87.194.dynamic.dol.ru (Dshield.org)
Also from a sans "Detects analyzed" document [1], this site was involved in malicious activity in the past.

Correlations:
This alert was studied by the following students also in their practical:
Lorraine Weaver[2] and Brian Credeur[3].

Defensive recommendations:
Closely follow or completely block traffic to and fro 194.86/16 network addresses.

Reference:
[1] www.sans.org/y2k/072818.htm
[2] Lorraine Weaver. http://www.giac.org/practical/Lorraine_Weaver_GCIA.zip
[3] Brian_Credeur, www.giac.org/practical/Brian_Credeur_GCIA.doc

Alert #4:SMB Name Wildcard
Alert-frequency – 29382 Category: Reconnaissance

The latest snort ruleset does not have a rule for this alert. The signature used may be similar as given below (this was obtained from a Google search.)

```
alert udp any any -> HOME_NET 137 (msg:"SMB Name Wildcard";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)
```

The signature is to detect 'wildcard' node status queries (or name query) on the network. The attacker can get information about all the NetBIOS names, which is there in the name table of the target. [1]

```
The part of RFC 1002 [2], which describes how a node status request is
shown. Note that a node status request with '*' is processed.
NODE STATUS REQUEST:
       /*
        * Name of "*" may be used for force node to
        * divulge status for administrative purposes
        */
        IF name in local name table OR name = "*" THEN
        BEGIN
            /*
             * Build response packet and send to requestor node
             * Send only those names that are in the same scope
             * as the scope in the request packet.
             */
           send NODE STATUS RESPONSE;
        END
```

The attacker might be looking for machines with unprotected sharing on drives. Besides, there are also worms which makes use of this method; for e.g. "911" bat-chode virus/worm and network.vbs worm.

A good connection between the pattern "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|" and SMB wildcard is given by Judy Novak in the following document [6].

```
Repeating from the document,
When NetBIOS names are sent over the network, they are "mangled". What
happens with an nbstat query is that :
Each character in NetBIOS name is divided into two hex characters
Normally blank padded to 16 characters. Each hex character added to
ASCII value 0x41 (uppercase "A") Now, if a wildcard name is used "*"
(as is done with the nbtstat –A IP address command), the formula is a
little different:
Each character in NetBIOS name is divided into two hex characters
Null padded to 16 characters. Each hex character added to ASCII value
0x41 (uppercase "A"). The value of "*" in hex is 2A. So the formula is
as follows
2 A (divided into two hex characters)
2 A (null padded – no change)
   2  A
+ 41 41 41 41 41 41 41 41 41 41 41 41 41 41, etc.
------------------------------------------------
  43 4B 41 41 41 41 41 41 41 41 41 41 41 41 - Hex result
   C  K  A  A  A  A  A  A  A  A  A  A  A  A - ASCII result
```

Correlation:
Bryce Alexander has analyzed similar alert in his GCIA practicals.[4]
He has also posted a “honey pot” catch to GIAC.[5]

Top talkers:

192.168.5.2	61.144.129.210	200.204.181.16	219.65.193.96	217.97.64.149

192.168.5.2
192.168.5.2 seems to be an internal machine and there is no evidence of a scan involved. Most probably this machine is involved in legitimate queries.

61.144.129.210
61.144.129.210 seems to be trying to get information using wildcard requests (*). See the traffic pattern from 61.144.129.210 given below. The attacker seems to be trying each IP address one by one as part of the reconnaissance.

```
01/17-07:51:00.686606  [**] SMB Name Wildcard [**]  61.144.129.210:1035 -> MY.NET.132.29:137
01/17-07:51:00.840997  [**] SMB Name Wildcard [**]  61.144.129.210:1035 -> MY.NET.132.30:137
01/17-07:51:00.988529  [**] SMB Name Wildcard [**]  61.144.129.210:1035 -> MY.NET.132.31:137
..
01/17-07:51:51.204424  [**] SMB Name Wildcard [**] 61.144.129.210:1035 -> MY.NET.133.43:137
01/17-07:51:51.361621  [**] SMB Name Wildcard [**] 61.144.129.210:1035 -> MY.NET.133.44:137
01/17-07:51:51.523226  [**] SMB Name Wildcard [**] 61.144.129.210:1035 -> MY.NET.133.45:137
```

These alerts can be correlated with **'SMB C access'** alerts from the same source IP address.

```
01/17-07:51:04.933227  [**] SMB C access [**]  61.144.129.210:4314 -> MY.NET.132.43:139
01/17-07:51:35.368758  [**] SMB C access [**]  61.144.129.210:4314 -> MY.NET.132.43:139
01/17-07:52:05.598991  [**] SMB C access [**]  61.144.129.210:4314 -> MY.NET.132.43:139
```

From the alerts it seems like MY.NET.132.43 responded to the initial probe and the attacker followed up by trying to access/mount the C directory.

The above alert entries can be correlated with entries from scan file also.
It has been logged as a UDP scan.

```
Jan 17 07:51:51 61.144.129.210:1035 -> 130.85.133.42:137 UDP
Jan 17 07:51:51 61.144.129.210:1035 -> 130.85.133.43:137 UDP
Jan 17 07:51:51 61.144.129.210:1035 -> 130.85.133.44:137 UDP
Jan 17 07:51:51 61.144.129.210:1035 -> 130.85.133.45:137 UDP
```

A 'whois' lookup from dshield.org on 61.144.129.210 gave:

Whois: Inetnum: 61.144.0.0 - 61.144.255.255
netname: CHINANET-GD
country: CN
descr: CHINANET Guangdong province network

200.204.181.16
A snippet from the alerts:

```
01/17-18:50:27.173074  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.95:137
01/17-18:50:27.365311  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.96:137
01/17-18:50:27.502245  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.97:137
01/17-18:50:27.708274  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.98:137
01/17-18:50:27.841784  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.99:137
01/17-18:50:28.030840  [**] SMB Name Wildcard [**] 200.204.181.16:1030 -> MY.NET.135.100:137
..
```

Obviously, the source is doing a reconnaissance work here, trying the 'wildcard request' on each machine on the target network.
A whois lookup on 200.204.181.16 at Dshield.org gave:

HostName:	200-204-181-16.speedyterra.com.br
Country:	
Total Records against IP:	2592
Number of targets:	2589
Date Range:	2003-01-10 to 2003-02-20

Ports Attacked (up to 10):

Port	Attacks
137	930

Defense recommendations:

1. Scanning for unprotected shares should be done at least once a quarter [7]
2. If file sharing is not needed disable Windows networking shares or consider disabling NetBIOS over TCP/IP altogether in the Windows network control panel. [8]

3. Block inbound and outbound NetBIOS traffic (TCP and UDP ports 137 through 139 and 445) on the perimeter firewall if NetBIOS is used.[8]

References:

[1] http://secinf.net/misc/An_analysis_of_TCPIP_NetBIOS_filesharing_protocols_.html
[2] Protocol standard for a NetBIOS service on a TCP/UDP transport: http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1002.html
[3] Port 137 Scan (Intrusion Detection FAQ) by Bryce Alexander http://www.sans.org/resources/idfaq/port_137.php
[4] Bryce Alexander, http://www.giac.org/practical/Bryce_Alexander.doc
[5] Follow up on a Honey Pot catch, Bryce Alexander - http://www.sans.org/y2k/honeypot_catch.htm
[6] Analysis by Judy Novak at GIAC. http://www.sans.org/y2k/061500.htm
[7] Bob Konigsberg, http://www.sans.org/rr/audit/inside.php
[8] David Cieslak, http://www.giac.org/practical/David_Cieslak_GSEC.doc

Alert #5:TFTP - External UDP connection to internal tftp server

Alert-frequency – 20698 Category: Illegal access/ Reconnaissance

TFTP or Trivial file transfer protocol is a simple protocol, running over UDP layer, and is used mainly by diskless machines and routers to download initial boot software from central servers. It does not support password authentication and has insecurity associated with it. As mentioned in the article – “Keeping Your Network Safe And Sound,

> It's easy to overlook the security of a device that's not on the front line of Internet access, but rather holds the configuration of the devices that connect you to the Internet. Viewing the device configuration on a TFTP server that is used to store network device configuration can be just as helpful to a hacker as viewing the configuration from the device itself.

Snort signature used might be –

```
alert udp any any -> HOME_NET 69
(msg:" TFTP - External UDP connection to internal tftp server";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)

alert udp HOME_NET 69 -> any any
(msg:" TFTP - External UDP connection to internal tftp server";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)
```

20692 out of 20698 alerts of the type as given below –

```
01/16-00:06:02.745396  [**]
TFTP - External UDP connection to internal tftp server
[**] MY.NET.111.231:69 -> 192.168.0.253:7933
01/16-00:06:02.745881  [**]
TFTP - External UDP connection to internal tftp server
[**] MY.NET.111.230:69 -> 192.168.0.253:7933
```

According to RFC 1918 [2], 192.168.0.253 belongs to the private address space. Snort thinks it to be an external IP address. Most probably, the HOME_NET variable might be configured to be MY.NET/16. It could have been configured to include the private addresses as well, in case there is NAT-ing being done and IDS is sitting inside the Firewall. The traffic seems normal.
However, there are 4 entries where the private addresses are not the case;

```
01/19-22:25:38.115465  [**]
TFTP - External UDP connection to internal tftp server [**]
63.210.198.194:2205 -> MY.NET.27.210:69
01/19-22:25:38.121441  [**]
TFTP - External UDP connection to internal tftp server [**]
63.210.198.194:2205 -> MY.NET.27.210:69
<snip>
```

It should be checked if MY.NET.27.210 is hosting a tftp server, and whether it needs to be accessible to outside world.
A whois on 63.210.198.194 gives

```
OrgName:    Level 3 Communications, Inc.
OrgID:      LVLT
```

Correlations:
Joe Ellis has done a brief description about this alert in his practical [3].

Recommendations:
1. It is a good practice to disable TFTP, if not needed. Otherwise, it should be made sure that only intended files are accessible. [1]
2. It is recommended that TFTP (UDP port 69) be blocked at the firewall.
3. Consider private addresses also in HOME_NET variable of snort configuration so as to reduce false alarms.

Reference:
[1] Network Security for Trade Shows; http://www.faqs.org/rfcs/rfc2179.html
[2] RFC 1928 http://www.isi.edu/in-notes/rfc1918.txt
[3] Joe Ellis, http://www.giac.org/practical/Joe_Ellis_GCIA.doc
[4] http://www.networkcomputing.com/818/818buyers3.html

Alert #6:spp_http_decode: IIS Unicode attack detected
Alert-frequency – 17819 Category: Illegal access, remote exec.

This is an attack against the Microsoft IIS servers. IIS 4.0 and 5.0 had "Web Server Folder Traversal" vulnerability, which gives an attacker illegal access to files and documents outside the webroot. It is also possible to remotely execute programs like cmd.exe, tftp.exe using this method. Andrew Brannan describes this attack very well in "Unicode Vulnerability – How & Why?" [1]

Microsoft IIS checks for the '../' pattern in the request, but using the Unicode encoding this check is bypassed. For e.g.
/ can be encoded as %c0%af , \ can be encoded as %c1%9c, . can be encoded as %2e.

This alert is generated by the HTTP_DECODE preprocessor of the snort. A look at the spp_http_decode.c (snort-1.8.6) reveals that the alert is generated when the preprocessor sees '%2f' or '%5c' or '%2e' in the URL.
There are also many worms based on this vulnerability - Nimda, Code Red, Sadmind to name a few.

Correlations:
This vulnerability is listed in cve database - CVE-2000-0884
Paul Critchfield has analyzed Unicode attack as part of his GCIA practical [2].

Top talkers:

Top Talkers (Internal)	Top Talkers (External)
MY.NET.88.249 (1293 alerts)	148.246.52.7
MY.NET.85.74 (1212 alerts)	211.90.88.43
MY.NET.84.133 (1140 alerts)	68.33.105.77
MY.NET.88.139 (875 alerts)	
MY.NET.153.110 (614 alerts)	

There is a high chance that these internal machines have been compromised (by some virus). More investigation on them is recommended.

Top targets:

207.200.86.97	211.233.32.56	199.244.218.42	MY.NET.70.207	MY.NET.168.140

Whether the attacks on the internal machines were successful is not evident. *However*, we can look for suspicious behavior from the internal servers for an indirect verification of a successful compromise.

MY.NET.70.207:
- 126 alerts: High port 65535 UDP - possible Red Worm – traffic

MY.NET.168.140:
- 20 alerts: Watchlist 000220 IL-ISDNNET-990517
- 3 alerts: Attempted Sun RPC high port access

```
01/18-20:49:51.600531  [**] Attempted Sun RPC high port access [**]
63.250.205.23:25142 -> MY.NET.168.140:32771
01/18-20:49:51.979193  [**] Attempted Sun RPC high port access [**]
63.250.205.23:25142 -> MY.NET.168.140:32771
01/18-20:49:56.428793  [**] Attempted Sun RPC high port access [**]
63.250.205.23:25142 -> MY.NET.168.140:32771
```

Information about the external top talkers:
148.246.52.7
Whois Information:
Hostname: ce590-gdl02.terra.net.mx
OrgName: TerraLycos Mexico
This external IP address has attempted the 'IIS Unicode attack', against a set of Web servers in the home network.

IIS Unicode attack attempt from 148.246.52.7			
Alert#	Destination IP	Alert#	Destination IP
215	MY.NET.157.52	152	MY.NET.130.122
197	MY.NET.130.40	149	MY.NET.130.34
191	MY.NET.132.42	129	MY.NET.150.220
184	MY.NET.136.2	127	MY.NET.130.123
171	MY.NET.150.228	114	MY.NET.130.86
170	MY.NET.157.12	97	MY.NET.157.11
167	MY.NET.130.91	95	MY.NET.198.241
158	MY.NET.137.18	75	MY.NET.198.237

211.90.88.43
Whois information:
Hostname: 211.90.88.43
inetnum: 211.90.0.0 - 211.91.255.255
netname: UNICOM
country: CN
descr: China United Telecommunications Corporation

68.33.105.77
Hostname : pcp02102752pcs.towson01.md.comcast.net
OrgName: Comcast Cable Communications, Inc.

False alarms is possible if a valid URL contains '%2f' or '%5c' or '%2e' and the only way to verify would be to have a look at the original packet payload. Snort IDS could be run in binary logging mode so that alerts based on content can be analyzed more.

Defensive recommendations:
1. Apply the necessary and recommended patches for the IIS web servers if any.
2. Moving the web folder root to a different logical drive than the one holding the system executables is a good idea. (given by Andrew Brannan in [1])

Reference:
[1] "Unicode Vulnerability – How & Why?" – Andrew Brannan
http://www.sans.org/rr/threats/unicode.php
[2] Paul Crutchfield, www.giac.org/practical/Paul_Crutchfield_GCIA.doc

Alert #7: High port 65535 udp - possible Red Worm - traffic
Alert-frequency - 4011 Category: Worm

The Red worm or Adore worm exploits vulnerabilities in rpc.statd, bind, LPRng, and wuftpd26. It creates a backdoor on the infected machines on port 65535. It is explained by Anthony Dell in "Adore Worm – Another Mutation" [1]. As per this document the backdoor is using TCP port 65535. I have not found any references about UDP port 65535 being used. However, traffic to port 65535 is suspicious. The snort rule that triggered must be similar to –

```
alert UDP any any -> any 65535 (msg:"High port 65535 udp - possible Red
Worm – traffic";)
alert UDP any 65535 -> any any (msg:"High port 65535 udp - possible Red
Worm – traffic";)
```

While analyzing these alerts it was noted that approximately 3400 out of 4000 alerts of this type was between port 6257 and port 65535.
Port 6257 (UDP) is used by a peer file sharing application called WinMX[2].

For e.g. a snippet of traffic from/to MY.NET.70.176 is given below:

```
01/16-00:16:12.167706  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.70.176:6257 -> 151.37.50.252:65535
01/16-00:18:00.469369  [**] High port 65535 udp - possible Red Worm - traffic [**]
151.37.50.252:65535 -> MY.NET.70.176:6257
01/16-00:22:56.265283  [**] High port 65535 udp - possible Red Worm - traffic [**]
151.37.50.252:65535 -> MY.NET.70.176:6257
<snip>
Corresponding entries in scans logs are :
Jan 16 00:16:12 130.85.70.176:6257 -> 151.37.50.252:65535 UDP
Jan 16 00:22:56 130.85.70.176:6257 -> 151.37.50.252:65535 UDP
Jan 16 00:22:59 130.85.70.176:6257 -> 151.37.50.252:65535 UDP
<snip>
```

From scans log files we can also note traffic (a snippet is shown below), which suggests that MY.NET.70.176 or 130.85.70.176 might be using WinMX.

```
Jan 16 00:16:11 130.85.70.176:6257 -> 65.28.231.53:6257 UDP
Jan 16 00:16:11 130.85.70.176:6257 -> 62.211.221.211:6257 UDP
Jan 16 00:16:11 130.85.70.176:6257 -> 68.58.122.61:6257 UDP
Jan 16 00:16:13 130.85.70.176:6257 -> 80.236.71.158:6257 UDP
Jan 16 00:16:12 130.85.70.176:6257 -> 162.83.151.188:6257 UDP
Jan 16 00:16:12 130.85.70.176:6257 -> 62.211.233.149:6257 UDP
Jan 16 00:16:12 130.85.70.176:6257 -> 24.27.252.13:6257 UDP
```

Top talkers:

MY.NET.70.176	211.125.217.3	MY.NET.83.146	MY.NET.150.213	MY.NET.91.72

Recommendations:

It is recommended that the internal machines are checked for possible infection / compromise.

Reference:

[1] Adore Worm – Another Mutation - J. Anthony Dell
http://www.sans.org/rr/threats/mutation.php
[2] WinMX - http://www.winmx.com

Alert #6: Incomplete Packet Fragments Discarded

Alert-frequency - 2863 Category: False alarm?

This is an alert generated from the fragmentation reassembly preprocessor, spp_defrag of the snort IDS. This is a deprecated preprocessor and should be replaced by the newer fragmentation reassembly preprocessor – spp_frag2.

This particular alert indicates that the fragment set is not complete. This may be due to lost fragments due to

- Network problems (lost fragments)
- IDS dropping packets
- Fragmentation attacks or
- False alarm. (described later)

After analyzing the spp_defrag.c, we see that the preprocessor does a check when it receives the last fragment of a set. If the total size of all received fragments is less that half the total length (of the original datagram before fragmentation) this alert is generated. Also this is checked only if total length > 8192 bytes. This is explained by Dragos Ruiu [7] and snippet of the mail is given below:

> This message is given by the defragmentation preprocessor when
> packets bigger than 8k that are more than half empty when the last
> fragment is received are discarded. This can be caused by:
> transmission errors, broken stacks, and fragmentation attacks So your assumption was correct.
> Hope it helps... and let me know if I can be of assistance. cheers,
> --dr

But this checking can cause false alarms especially with O.S's like Linux which sends fragments out in the opposite order; i.e last fragment is sent out first. I came up with this scenario, which caused the IDS to throw the (false) alert given below –

The fragment set which triggered the alert is also given below. As can be noticed the fragments are generated from a Linux machine and therefore is seen on the wire in the opposite order (last fragment first).

```
[**] [103:2:1] Incomplete Packet Fragments Discarded [**]
02/06-15:14:46.869357 192.168.16.55 -> 192.168.16.54
ICMP TTL:64 TOS:0x0 ID:34780 IpLen:20 DgmLen:9028
ICMP header truncated
```

```
15:14:46.869357 192.168.16.55 > 192.168.16.54: (frag 34780:128@8880)
```

```
15:14:46.869371 192.168.16.55 > 192.168.16.54: (frag 34780:1480@7400+)
15:14:46.869385 192.168.16.55 > 192.168.16.54: (frag 34780:1480@5920+)
15:14:46.869393 192.168.16.55 > 192.168.16.54: (frag 34780:1480@4440+)
15:14:46.869401 192.168.16.55 > 192.168.16.54: (frag 34780:1480@2960+)
15:14:46.869409 192.168.16.55 > 192.168.16.54: (frag 34780:1480@1480+)
15:14:46.869417 192.168.16.55 > 192.168.16.54: icmp: echo request (frag
34780:1480@0+)
```

Also, this alert is generated only for fragment sets whose reassembled size is greater than 8192 bytes.

Correlations:
John Jenkinson has discussed this alert in his practical [3].

Recommendations:
- Upgrade the fragmentation reassembly preprocessor. This is suggested by Martin Roesche in one of the mails in snort archive [2].
- Keep an eye whether the IDS is dropping packets. In case it is dropping packets, then the configuration can be tuned so that the load on the IDS is reduced.

References:
[1] http://www.security-express.com/archives/snort/2001-02/0320.html
[2] http://archives.neohapsis.com/archives/snort/2001-11/0822.html
[3] John Jenkinson, http://www.giac.org/practical/John_Jenkinson_GCIA.doc

Alert #9:Possible trojan server activity
Alert-frequency – 2688 Category: Trojan

Port 27374 is associated with many Trojan/worm applications; mainly subseven Trojan, Lion and ramen worm. [1]

2491 out of 2688 alerts were triggered by traffic between MY.NET.91.104 and 213.46.21.207. A small snippet of the traffic is shown:

```
01/17-02:38:11.325953  [**] Possible trojan server activity [**] MY.NET.91.104:1214 -> 213.46.21.207:27374
01/17-02:38:18.915710  [**] Possible trojan server activity [**] 213.46.21.207:27374 -> MY.NET.91.104:1214
01/17-02:38:20.347762  [**] Possible trojan server activity [**] MY.NET.91.104:1214 -> 213.46.21.207:27374
01/17-02:38:20.347892  [**] Possible trojan server activity [**] MY.NET.91.104:1214 -> 213.46.21.207:27374
01/17-02:38:20.352029  [**] Possible trojan server activity [**] 213.46.21.207:27374 -> MY.NET.91.104:1214
```

1214 is the port used by KaZaa. It could be very probable that this traffic is non malicious since 27374 is an ephemeral port and there is a certain chance that nodes are going to use it for legitimate purposes.

Most probably, the snort rule used for this alert must be:

```
alert TCP any any -> any 27374 (msg:"Possible trojan server activity";)
```

```
alert TCP any 27374 -> any any (msg:” Possible trojan server activity;)
```

A rule as this one can create false alarms, and it is recommended to use signatures based on content of the packet. The latest snort ruleset has the following rule for Subseven Trojan:

```
alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR subseven 22";
flow:to_server,established; content:"|0d0a5b52504c5d3030320d0a|"; reference:arachnids,485;
reference:url,www.hackfix.org/subseven/; classtype:misc-activity; sid:103; rev:5;)
```

The top talkers of this alert, as mentioned above, are MY.NET.91.104 and 213.46.21.207.
A lookup on 213.46.21.207 at Dshield.org gives –
Hostname: d21207.upc-d.chello.nl

Recommendations:
Use better IDS signatures with some specific content matching with the trojan’s signature; for example, from the latest snort rule set.

References:
[1] http://isc.incidents.org/port_details.html?port=27374

Alert #10:spp_http_decode: CGI Null Byte attack detected
Alert-frequency - 1900 Category: Web exploit

As the alert shows, this is generated by the ‘HTTP Decode’ preprocessor of the Snort IDS. As seen from spp_http_decode.c (snort source code) and also from Snort FAQ [1], this alert is generated when a %00 is seen while doing http decoding. As the FAQ also points out, there is a likely chance of false alarms, especially when sites use cookies. This might reason well with the large number of alerts of this type. Another reason is given below-

Take a look at the following trait. Look at the timestamp of each set of alerts.

```
01/16-11:24:28.632236  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.27.231:2048 -> 216.241.219.12:80
01/16-11:24:28.632236  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.27.231:2048 -> 216.241.219.12:80
<repeated many times>

01/16-11:24:39.239788  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.27.231:2048 -> 216.241.219.12:80
01/16-11:24:39.239788  [**] spp_http_decode: CGI Null Byte attack
detected [**] MY.NET.27.231:2048 -> 216.241.219.12:80
<repeated many times>
```

This pattern is there through out for this alert. The http decoder goes through each character of the request uri and does the check for %00. So if one request itself has more than one %00, the alert is generated that many times.

This can be corrected by removing the exactly same lines using the 'sort' and 'uniq' commands of unix. After doing so, the frequency of the alerts came down from 1900 to 569.

Top talkers:

MY.NET.117.150	MY.NET.117.149	MY.NET.168.105	MY.NET.53.67	MY.NET.115.229

As Snort FAQ suggests, the best way to verify if we are handling a real attack packet is to have the real packet dump. It is possible to run snort to log the actual packet in case there is an alert generated for the particular packet. (We do not have this for this assignment)

Correlations:
Bradley Urwiller has discussed this alert in his GCIA practical. [2]

Recommendations:
Run Snort with –b option to capture the packet which triggers the alert. This will enable to verify alerts that need further analysis.

Reference:
[1] Snort FAQ, http://www.snort.org/docs/faq.html#4.12
[2] Bradley Urwiller, http://www.giac.org/practical/Bradley_Urwiller_GCIA.pdf

Alert #11:EXPLOIT x86 NOOP	
Alert-frequency - 1757	Category: Exploit

NOOPs are used to increase the success chances of buffer overflow attempts; this is described in 'Intrusion Signatures and Analysis' textbook (Detect by Mark Cooper, GCIA). Reciting from his practical:

```
Because it is very difficult for the attacker to know exactly where in
memory the rogue code resides, and thus what value must be placed into
the return pointer, the rogue code is surrounded by a large number of
NO-OP instructions.
<snip>
The hex code for the NO-OP instruction on the Intel x86 family of
processors is 0x90. Therefore, the IDS is programmed to look at the
content of packets for repeated instances of the byte 0x90.
```

The snort rule used might have been

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP";
content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90|"; flags: A+; reference:arachnids,181;)
```

Just looking at the signature itself, we can say that there are good chances of false alarms. One example is described by Chris Keuthe in his practicals [1] The packet happened to contain some jpeg image !

Top talkers:

80.13.63.154	217.128.167.158	80.14.125.161	164.106.62.52	24.189.153.176

Correlations:

GCIA detect – Bryce Alexander (Intrusion Signatures and analysis)
Mark Cooper (Intrusion Signatures and analysis)

Reference:

[1] Chris Kuethe, http://www.giac.org/practical/chris_kuethe_gcia.html#1.1

Alert #12: IDS552/web-iis_IIS ISAPI Overflow ida nosize	
Alert-frequency – 1607	Category: Exploit

Microsoft IIS web server version 4.0 and 5.0 are vulnerable to a buffer overflow attack due to problems in the handling of ISAPI (Internet Services Application Programming Interface) extensions. [1] This could potentially be taken advantage of by an attacker to gain control of a vulnerable web server. Code Red and Code Red II worms make use of this vulnerability.

A web request of the form below is used [3]

```
GET /NULL.ida?[buffer]=X HTTP/1.1
Host: werd
Where [buffer] is aprox. 240 bytes.
```

For e.g. [4]

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%
u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531
b%u53ff%u0078%u0000%u00=a
```

Top talkers:

140.128.19.195	140.128.19.224	203.145.177.111	64.221.193.90	212.209.9.218

Correlations:

The vulnerability can be found at cve.mitre.org with id - CAN-2001-0500 [2]

Defensive recommendations:

- Update or apply relevant patches for the IIS web servers.

Reference:

[1] http://www.iss.net/security_center/static/6705.php
[2] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0500
[3] http://www.eeye.com/html/Research/Advisories/AD20010618.html
[4] http://www.cert.org/advisories/CA-2001-19.html

Alert #13 : Queso fingerprint
Alert-frequency - 1573 Category: reconnaissance

Summary: 'Queso' is a probe used for OS finger printing. Queso is the Hispanic shortcut for "Que Sistema Operativo?" which translates into "which operating system?" [3]. Among other types of scans Queso does, one method (BOGUS flag probe) is by setting the reserved bits of TCP flags with a SYN segment to fingerprint the target machine's O.S.

Toby Miller has given a discussion at SANS on 'ECN and it's impact on Intrusion Detection' – which is very useful in this analysis [2]. The recent ECN standard uses the reserved bits in TCP header and IP header for ECN – Explicit Congestion Notification. (More discussion below)
Fyodor describes the reconnaissance method more in the article Remote OS detection via TCP/IP Stack FingerPrinting [1].

> The BOGUS flag probe -- Queso is the first scanner I have seen to use this clever test. The idea is to set an undefined TCP "flag" (64 or 128) in the TCP header of a SYN packet. *Linux boxes prior to 2.0.35* keep the flag set in their response. I have not found any other OS to have this bug. However, some operating systems seem to reset the connection when they get a SYN+BOGUS packet. This behavior could be useful in identifying them.

The latest snort rule set did not have the rule for this scan. But the appropriate signature would be:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" Possible Queso
Fingerprint attempt";flags:S12;)
```

Alex Stevens (see correlation section) has listed the above snort rule.
As expected, the SCAN files have corresponding entries as shown below; and it can be seen that the SYN flag and the reserved bits '1' and '2' are also set.

```
01/16-22:36:52.861440  [**] Queso fingerprint [**] 65.214.36.150:59054
-> MY.NET.99.174:80
01/16-23:01:21.375013  [**] Queso fingerprint [**] 65.214.36.150:39848
-> MY.NET.99.85:80

Jan 16 22:36:52 65.214.36.150:59054 -> 130.85.99.174:80 SYN 12****S*
RESERVEDBITS
Jan 16 23:01:21 65.214.36.150:39848 -> 130.85.99.85:80 SYN 12****S*
RESERVEDBITS
```

The part of the TCP header is shown below: (the 13th - 16th bytes)

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Data  |           |U|A|P|R|S|F|                                   |
   | Offset| Reserved  |R|C|S|S|Y|I|            Window                 |
   |       |           |G|K|H|T|N|N|                                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RFC 793 speaks about the 'Reserved' bits as follows -

Reserved: 6 bits
Reserved for future use. Must be zero.

ECN and Queso:
The use of reserved bits of the tcp flags for ECN has complicated the detection of Queso fingerprint attempts. The point to note is that if the packet in question is a valid ECN packet, then besides the reserved bits of tcp flags, the 6th bit of TOS filed in IP header will also be set. [2] [4]

Top talkers:

65.214.36.150	217.126.116.244	66.140.25.156	81.56.17.119	195.71.116.19
202.156.131.251	209.47.251.30	209.47.251.23	209.47.251.25	209.167.239.31

Let's consider some of the top talkers for further analysis.
65.214.36.150:
nslookup 65.214.36.150 : egspd400.teoma.com
Look up at www.arin.net:
AskJeeves, Inc. UU-65-214-36 (NET-65-214-36-0-1)
65.214.36.0 - 65.214.39.255

```
01/19-17:16:06.362703  [**] Queso fingerprint [**] 65.214.36.150:43380
-> MY.NET.162.87:80
01/19-17:06:54.260021  [**] Queso fingerprint [**] 65.214.36.150:50038
-> MY.NET.140.2:80
01/19-17:25:56.484936  [**] Queso fingerprint [**] 65.214.36.150:53209
-> MY.NET.130.123:80
01/19-17:48:09.880586  [**] Queso fingerprint [**] 65.214.36.150:59216
-> MY.NET.150.83:80
01/19-18:22:04.163839  [**] Queso fingerprint [**] 65.214.36.150:51577
-> MY.NET.145.18:80
<snip>
```

On first look although it seems that this host is doing a Queso scan on the destination network, most probably 65.214.36.150 is doing some kind of "spider" like functionality as search engines do. These are most probably false alarms.

209.47.251.30 : smtp20.rapid-e.net
209.47.251.23 : smtp13.rapid-e.net
209.47.251.25 : smtp15.rapid-e.net

These seems to be having legitimate SMTP conversations with MY.NET.6.40 For example, there are 32 alerts of Queso scan from 209.47.251.30 to MY.NET.6.40 and 26 alerts from 209.47.251.23. Normally, there is no reason an attacker would 32 probes to do some O.S fingerprinting. These are most probably false alarms.

Correlations:
Bryce Alexander discusses 'Queso' packets in his network detects (SF set in Header). He discusses about how to recognize a Queso packet from a tcpdump log:

```
The hex value c2 in the Tcpdump trace can be analyzed using figure
17.1. Since hex c is the decimal for 12, we know that the bits
representing 8 and 4 must be set; these correspond to the two reserved
bits. The hex value 2 corresponds to Syn flag. What makes the use of
reserved bits "stealthy" is that if an analyst were to use TCPdump to
view the trace, the reserved bits would not be seen unless the packet
hex dumps were closely examined.
```

The latest tcpdump, however does indicate the setting of the reserved bits as shown below. Note the flags "SWE"

```
13:22:02.551094 130.207.16.55.1777 > 130.207.16.54.0: SWE
1621615244:1621615244(0) win 512
0x0000   4500 0028 6841 0000 4006 ec83 82cf 1037        E..(hA..@......7
0x0010   82cf 1036 06f1 0000 60a7 e28c 5351 49e7        ...6....`...SQI.
0x0020   50c2 0200 9fb9 0000                            P.......
```

Correlations:
This alert is described by Alex stevens in his Network Detects section. [5]

Recommendations:
A better IDS signature to reduce false alarms could be used. I have not tried it, but the rule can be tried.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" Possible Queso
Fingerprint attempt";flags:S12; tos:0x2; )
```

References:

[1] Fyodor, http://www.insecure.org/nmap/nmap-fingerprinting-article.txt
[2] ECN and it's impact on Intrusion Detection - Toby Miller
[3] http://leb.net/hzo/ioscount/
[4] A Proposal to add Explicit Congestion Notification (ECN) to IP, RFC 2481
[5] Alex Stevens, http://www.giac.org/practical/Alex_Stephens_GCIA.htm

Alert #14: SUNRPC highport access!	
Alert-frequency - 1227	Category: Reconnaissance

Rpcbind or portmapper is a server which is needed to run Remote procedure calls. They convert RPC program numbers to universal addresses or port numbers. On Solaris, this program listens on port 32771, rather than the conventional port 111 [1]. So if the packet filter or firewall is not aware this port, it maybe open at the firewall.

RPC weaknesses is listed as number 3 in the top 10 Internet security threats [2]

The document [2] also says –

```
There is compelling evidence that the vast majority of the
distributed denial of service attacks launched during 1999 and
early 2000 were executed by systems that had been victimized
because they had the RPC vulnerabilities.
```

Correlations:
There are a number of vulnerabilities associated with RPC. The CVE entries related to a few of them are:
CVE-1999-0687 - The ToolTalk ttsession daemon uses weak RPC authentication, which allows a remote attacker to execute commands.
CVE-1999-0003 - Execute commands as root via buffer overflow in Tooltalk database server (rpc.ttdbserverd)
CVE-1999-0696 – Buffer overflow in rpc.cmsd

Top talkers	Details
128.122.20.14	SLINKY.CS.NYU.EDU
66.250.223.22	Cogent Communications
66.35.250.209	projects.sourceforge.net

Recommendations:
- Turn off the RPC related services on hosts, if not needed.
- Block ports 111 and 32771 at the firewall.
- Apply latest patches for the machines running rpc related applications.

Reference:
[1] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0189
[2] http://www.sans.org/top20/top10.php

Alert #15: TCP SRC and DST outside network
Alert-frequency – 906 Category: Abnormal

This alert notifies about packets having source IP address and destination IP address of outside Internet. Such packets should not be seen by IDS normally. The alerts are due to

1. IDS considering private addresses as outside address,
2. Mis-configurations in IP addresses.
3. Possible spoofing from internal machine.

The rule used for this alert could have been:

```
Set EXTERNAL_NET !HOME_NET

alert tcp $EXTERNAL_NET any -> $EXTERNAL_NET any (msg:" TCP SRC and DST
outside network"; flags:S12; tos:0x2; )
```

597 out of 906 alerts of this type are due to the fact that IDS triggers the alert interpreting the private address 192.168* as external IP address. The HOME NETWORK seems to be be configured as MY.NET/16. A snippet is shown below:

```
01/16-09:33:25.649780  [**] TCP SRC and DST outside network [**]
192.168.0.147:4829 -> 208.240.243.36:80
01/16-09:35:14.544070  [**] TCP SRC and DST outside network [**]
192.168.0.147:4836 -> 216.49.88.100:80
```

Some alerts have both private addresses. This traffic must have been captured by an IDS listening to internal traffic. (Information regarding IDS placement is not available.)

```
01/16-19:20:29.208724  [**] TCP SRC and DST outside network [**]
192.168.3.4:139 -> 192.168.2.5:1228
01/16-19:20:33.146058  [**] TCP SRC and DST outside network [**]
192.168.3.4:139 -> 192.168.2.5:1228
01/17-08:50:16.843280  [**] TCP SRC and DST outside network [**]
192.168.1.147:49154 -> 192.168.1.46:139
```

The above alerts are false alarms and can be avoided by setting the HOME_NET variable to include the private addresses as well.
Now, let us take a look at the other alerts, where private addresses were not the case. Take a look at this set of alerts:

```
01/16-10:20:12.152097  [**] TCP SRC and DST outside network [**] 192.2.3.11:1091 -> 192.4.3.11:12865
01/16-10:10:41.540589  [**] TCP SRC and DST outside network [**] 192.2.3.11:1088 -> 192.4.3.11:12865
01/16-10:10:43.551819  [**] TCP SRC and DST outside network [**] 192.4.3.11:12865 -> 192.2.3.11:1088
01/16-10:21:16.166902  [**] TCP SRC and DST outside network [**] 192.2.3.11:1091 -> 192.4.3.11:12865

01/19-20:37:14.421575  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32843
01/19-20:37:15.045889  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32843
01/19-20:37:17.565481  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32843
01/19-20:37:27.644275  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32843

01/20-12:23:45.474534  [**] TCP SRC and DST outside network [**] 192.5.3.11:32863 -> 192.6.3.11:1720
01/20-12:23:46.095149  [**] TCP SRC and DST outside network [**] 192.5.3.11:32863 -> 192.6.3.11:1720
01/20-12:23:48.614913  [**] TCP SRC and DST outside network [**] 192.5.3.11:32863 -> 192.6.3.11:1720
01/20-12:23:53.007186  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32863
01/20-12:23:54.267028  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32863
```

This is interesting. Note the IP addresses and their details from www.arin.net

IP Address	Organization
192.2.3.11	Genuity
192.4.3.11	Telcordia Technologies
192.6.3.11	Agilent Technologies
192.5.3.11	City of Beverly Hills

It is strange to see traffic from Genuity to Telcordia or from Agilent to City of Beverly hills in this network.

- Most probably this is due to some mis-configurations in IP addresses allocation. Some internal machines are using non private IP addresses and are captured by IDS listening on the internal network. 192.168/16 is the private address space to be used and not 192/24.
- What are the chances of IP address spoofing ?
  Take for e.g. the two alerts –

```
01/20-12:23:48.614913  [**] TCP SRC and DST outside network [**] 192.5.3.11:32863 -> 192.6.3.11:1720
01/20-12:23:53.007186  [**] TCP SRC and DST outside network [**] 192.6.3.11:1720 -> 192.5.3.11:32863
```

In a usual IP address spoofing case, if one of the internal machine spoofs the source IP address as 192.5.3.11 and sends a packet to 192.6.3.11, we will not see a reply. Why ? Because 192.6.3.11 will be sending the reply to the original 192.5.3.11. But we do see the reply. So the only way this could happen is that both the machines are in the internal network.

A last set of alerts is shown below:

```
01/16-09:35:38.844958  [**] TCP SRC and DST outside network [**] 169.254.84.3:2110 ->
68.55.170.231:139
01/16-09:45:45.708323  [**] TCP SRC and DST outside network [**] 169.254.84.3:2123 ->
68.55.170.231:139
01/16-09:46:56.875805  [**] TCP SRC and DST outside network [**] 169.254.84.3:2126 ->
68.55.170.231:139
01/16-09:46:59.820301  [**] TCP SRC and DST outside network [**] 169.254.84.3:2126 ->
68.55.170.231:139
....
```

This is most probably due to address spoofing by an internal machine; and IP address mis-configuration is also a possibility.

Defensive recommendation:
Egress filtering on the routers/firewalls to eliminate packets which have source IP address spoofing.

6. Overall Top talkers:

Considering the alerts from the five day period, the top 10 talkers in terms of number of alerts were –

IP Address	# of alerts	Type of alerts
MY.NET.84.151	26997	'Red worm (tcp)' alert.
MY.NET.105.204	22638	Most of the alerts are watchlist alerts; either Russia Dynamo or Watchlist 000220 IL-ISDNNET-990517
194.87.6.86	16545	All the alerts are 'Russia Dynamo'
212.179.1.145	16195	'Watchlist 000220 IL-ISDNNET-990517'
217.136.73.54	6773	Red Worm (tcp)
212.179.56.252	5889	'Watchlist 000220 IL-ISDNNET-990517'

MY.NET.111.235	4184	TFTP - External UDP connection to internal tftp server
MY.NET.111.232	4173	TFTP - External UDP connection to internal tftp server
MY.NET.111.219	4135	TFTP - External UDP connection to internal tftp server
MY.NET.111.231	4110	TFTP - External UDP connection to internal tftp server

7. OOS Analysis:

Out of specification or OOS packets are those that are abnormal according to the specifications of the protocol. For e.g. a TCP segment with the SYN and Fin flags turned on would fall in this category.

These kinds of packets are used for fingerprinting operating systems, evading intrusion detection systems and getting past firewalls. Bad routers also could create such abnormality [1].

Usage of the reserved bits of TCP header was out of spec, but these are used for ECN or Explicit congestion notification now. This makes detecting the actual malicious probe packet a bit difficult. This was discussed earlier with regards to Queso attempt.

TCP Flags related OOS:

The OOS files analyzed had 108 different combinations for TCP flags which are out of spec. The ones that occurred more than 5 times are listed below:

Type	Frequency		Type	Frequency
12****S* (Queso)	4989		1*U*PRSF	6
******** (Null)	967		1*UA*RSF	6
****P*** (VECNA)	648		1***PRSF	6
12***R**	26		12U**R*F	6
**U*P*SF	9		12U*PR**	6
12UA***F	9		12UA****	6
U***	8		12**PR*F	6
****PRSF	8		12*A****	6
*2UAPRSF	8		12******	6
12**P**F	8		**UAP*SF	5
1**A*RSF	7		*2UAP*SF	5
12**P*SF	7		12UA*RS*	5
12**P*S*	7		12UAPRSF	5
12*A*R*F	7		12U*****	5
******SF (Syn Fin)	6		12**P***	5

Fragmentation related OOS:

There were also 11 fragmentation related OOS packets. There can be different kind of OOS packets related to fragmentation. For.e.g. Tiny fragments. An example of tiny fragmentation attack is given in RFC 1858 [2] where the first fragment has just 8 bytes. This means that this will have the source port and destination port information but not the tcp flags. This method can evade firewalls

that do not perform fragmentation reassembly before passing the packets and some IDSs also.

The OOS alerts in the analyzed OOS files had the DF and MF bits set, which is not normal.

```
01/18-14:51:29.398891 65.42.92.54 -> MY.NET.153.210
TCP TTL:112 TOS:0x0 ID:3985 IpLen:20 DgmLen:752 DF MF
Frag Offset: 0x0   Frag Size: 0x2DC
01/18-14:51:29.935853 65.42.92.54 -> MY.NET.153.210
TCP TTL:112 TOS:0x0 ID:3991 IpLen:20 DgmLen:752 DF MF
Frag Offset: 0x0   Frag Size: 0x2DC
01/18-14:51:30.298141 65.42.92.54 -> MY.NET.153.210
TCP TTL:112 TOS:0x0 ID:3995 IpLen:20 DgmLen:752 DF MF
Frag Offset: 0x0   Frag Size: 0x2DC
01/18-14:51:30.385900 65.42.92.54 -> MY.NET.153.210
TCP TTL:112 TOS:0x0 ID:3996 IpLen:20 DgmLen:752 DF MF
Frag Offset: 0x0   Frag Size: 0x2DC
```

Top 10 talkers in terms of OOS alerts were –

IP Address	Number of alerts	Type
65.214.36.150	765	12****S*
209.191.132.40	632	12****S*
MY.NET.70.183	560	******** (Null scan)
148.63.115.208	558	****P***
217.126.116.244	487	12****S*
MY.NET.53.10	312	********
66.189.101.206	241	This IP address had 86 types of OOS alerts.
195.71.116.19	169	12****S*
66.140.25.156	165	12****S*
202.156.131.251	148	12****S*

Note that most of the OOS alerts is '12****S*'. This flag combination matches with Queso fingerprint. But the signature could be part of the Explicit Congestion Notification (ECN) scheme which uses the TCP reserved bits. This is discussed with the 'Queso' description earlier.

Reference:
[1] Chapter 17, Intrusion Signatures and Analysis – Northcutt et al.
[2] RFC 1858 - Security Considerations for IP Fragment Filtering

8. Scan analysis:

Top talkers

IP address	Number of entries
130.85.84.147	1272181
130.85.83.146	1001877
130.85.70.176	531468

130.85.150.213	282957
130.85.91.72	276771
130.85.114.45	231185
130.85.91.252	126497
130.85.88.242	102531
130.85.118.6	77679
130.85.198.220	73537

Top scanners (external)

IP address	Name resolution and details	Freq
68.33.105.77	pcp02102752pcs.towson01.md.comcast.net This machine is involved in a set of vertical scans on different targets. A snippet is shown below: Jan 17 01:48:06 68.33.105.77:2537 -> 130.85.70.75:1 SYN ******S* Jan 17 01:48:06 68.33.105.77:2538 -> 130.85.70.75:2 SYN ******S* Jan 17 01:48:06 68.33.105.77:2539 -> 130.85.70.75:3 SYN ******S* Jan 17 01:48:06 68.33.105.77:2540 -> 130.85.70.75:4 SYN ******S* …… Jan 17 01:48:07 68.33.105.77:2561 -> 130.85.70.75:23 SYN ******S* Jan 17 01:48:07 68.33.105.77:2562 -> 130.85.70.75:24 SYN ******S* Jan 17 01:48:07 68.33.105.77:2566 -> 130.85.70.75:25 SYN ******S*	17271
217.136.117.165	165.117-136-217.adsl.skynet.be This machine has actively scanned port 135 and port 80 on the network. However, this machine has not triggered any alerts.	16632
80.13.63.149	ABoulogne-112-1-1-149.abo.wanadoo.fr Scanning for port 135, 80 and 445 (which is used for SMB over TCP). No alerts other than portscan.	16434
217.128.46.154	AMontpellier-205-1-9-154.abo.wanadoo.fr Scanning for port 3389 (a total of 15569 SYNs covering almost the whole network). Port 3389 is associated with MS Terminal server. There are vulnerabilities on certain versions of the application and an attacker can execute hostile code on the server. [1]	15567
81.48.118.178	ALimoges-102-1-2-178.abo.wanadoo.fr Does a vertical scan on 130.85.150.210. From the logs we know that 130.85.150.210 runs a TFTP server.There are 4 alerts from this IP. 01/16-06:57:47.606153 [**] TFTP - External TCP connection to internal tftp server [**] 81.48.118.178:1932 -> MY.NET.150.210:69 01/16-06:57:47.606181 [**] TFTP - External TCP connection to internal tftp server [**] MY.NET.150.210:69 -> 81.48.118.178:1932 01/16-06:57:54.183046 <snip>	14449

Top 5 destination ports (according to scan reports)

Port number	Application
6257/UDP	WinMX program uses this port and is being extensively used through out the network. This port can be blocked at the firewall; but that is just a temporary solution since WinMX can be configured to use other ports.
135	NetBIOS
445	Used for SMB over TCP
80	HTTP
41170	Unknown

Some scanning activity by internal machines

- 130.85.82.32 was seen scanning 528 external IP addresses on port 80 between the period Jan 17th 3:22 and 4:36
- 130.85.190.90 was scanning external IP addresses for port 139. The activity on this port is described in ISS's doc as –

> This is the single most dangerous port on the Internet. All "File and Printer Sharing" on a Windows machine runs over this port. About 10% of all users on the Internet leave their hard disks exposed on this port. This is the first port hackers want to connect to, and the port that firewalls block.

Reference:

[1]
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-087.asp

9. Further investigation of 5 external IP addresses:

The top 15 external IP addresses in terms of most number of alerts are shown below. The top 5 are selected (since there are 5 IPs from the 212.179/16 network and they all triggered the same alert, one of them is taken as the representative).

1	**194.87.6.86**	6	212.179.105.69	11	148.246.52.7
2	**212.179.1.145**	7	212.179.98.160	12	61.166.111.117
3	**217.136.73.54**	**8**	**62.147.242.129**	13	212.95.85.172
4	212.179.56.252	**9**	**80.200.147.156**	**14**	**68.33.105.77**
5	212.179.107.228	10	217.225.205.66	15	80.200.147.21

Besides the top five, 68.33.105.77 is also selected due to a special consideration; i.e this IP address triggered the most number of unique alerts. They were -

Alert name	Frequency
TFTP - External TCP connection to internal tftp server	160
External POP to HelpDesk MY.NET.70.49	349

External POP to HelpDesk MY.NET.70.50	276
PHF attempt	12
spp_http_decode: IIS Unicode attack detected	410
External RPC call	113
External FTP to HelpDesk MY.NET.70.49	15
External FTP to HelpDesk MY.NET.70.50	14
spp_http_decode: CGI Null Byte attack detected	8

1. **194.87.6.86**
This IP address generated the most alerts (16545) and all were 'Russia Dynamo' alerts, which was discussed earlier.

nslookup 194.87.6.86

Non-authoritative answer:
86.6.87.194.in-addr.arpa name = **86.6.87.194.dynamic.dol.ru**.
Authoritative answers can be found from:
6.87.194.in-addr.arpa nameserver = ns.demos.su.
6.87.194.in-addr.arpa nameserver = ns1.demos.net.
ns.demos.su internet address = 194.87.0.8
ns.demos.su internet address = 194.87.0.9

Information from whois service at www.ripn.net

```
inetnum:        194.87.6.0 - 194.87.6.255
netname:        DEMOS-DOL-DIALUP
descr:          DEMOS-Online Dialup
descr:          Demos-Internet Co.
descr:          Moscow, Russia
country:        RU
admin-c:        DNOC-ORG
source:         RIPE
```

2. **212.179.1.145**

nslookup 212.179.1.145

Non-authoritative answer:
145.1.179.212.in-addr.arpa name = **fr-c27145.kbm.org.il**.
Authoritative answers can be found from:
1.179.212.in-addr.arpa nameserver = ns1.bezeqint.net.
1.179.212.in-addr.arpa nameserver = ns2.bezeqint.net.
ns1.bezeqint.net internet address = 192.115.106.10
ns2.bezeqint.net internet address = 192.115.106.11

Information from whois service at www.ripe.net

```
inetnum:        212.179.1.128 - 212.179.1.191
```

```
netname:        KIBBITZ-KFAR-BLUM
mnt-by:         INET-MGR
descr:          KIBBITZ-KFAR-BLUM-LAN
country:        IL
admin-c:        ZV140-RIPE
status:         ASSIGNED PA
notify:         hostmaster@isdn.net.il
changed:        hostmaster@isdn.net.il 20020902
address:        bezeq-international
source:         RIPE
```

3. **217.136.73.54**

nslookup 217.136.73.54

Non-authoritative answer:
54.73.136.217.in-addr.arpa name = 54.73-136-217.adsl.skynet.be.
Authoritative answers can be found from:
73.136.217.in-addr.arpa nameserver = ns4.skynet.be.

Information from www.ripe.net

```
inetnum:        217.136.0.0 - 217.136.127.255
netname:        BE-SKYNET-ADSL1
descr:          Belgacom Skynet SA/NV
descr:          ADSL Access
country:        BE
admin-c:        SN2068-RIPE
tech-c:         SN2068-RIPE
rev-srv:        ns.ripe.net
rev-srv:        ns1.skynet.be
mnt-by:         SKYNETBE-MNT
changed:        ripe@skynet.be 20021125
source:         RIPE
```

4. **62.147.242.129**

nslookup 62.147.242.129
Non-authoritative answer:
129.242.147.62.in-addr.arpa
name = lns-p19-25-62-147-242-129.adsl.proxad.net.
Authoritative answers can be found from:
242.147.62.in-addr.arpa nameserver = ns1.proxad.net.

Information from www.ripe.net

```
inetnum:        62.147.79.0 - 62.147.255.255
netname:        FR-PROXAD-DIALUP
descr:          Proxad / Free Telecom
descr:          Dynamic pool (dialup)
descr:          NCC#2002110278 (45312/45824)
```

```
country:        FR
admin-c:        ACP23-RIPE
tech-c:         TCP8-RIPE
status:         ASSIGNED PA
mnt-by:         PROXAD-MNT
source:         RIPE
```

5. **80.200.147.156**

nslookup 80.200.147.156

Non-authoritative answer:
156.147.200.80.in-addr.arpa name = 156.147-200-80.adsl.skynet.be.
Authoritative answers can be found from:
147.200.80.in-addr.arpa nameserver = ns4.skynet.be.

Information from www.ripe.net

```
inetnum:        80.200.0.0 - 80.200.255.255
netname:        BE-SKYNET-20011108
descr:          ADSL Customers
descr:          Skynet Belgium
country:        BE
admin-c:        JFS1-RIPE
tech-c:         PDH16-RIPE
status:         ASSIGNED PA
mnt-by:         SKYNETBE-MNT
changed:        ripe@skynet.be 20011212
source:         RIPE
```

6. **68.33.105.77**

nslookup 68.33.105.77
Non-authoritative answer:
77.105.33.68.in-addr.arpa name =
pcp02102752pcs.towson01.md.comcast.net.
Authoritative answers can be found from:
33.68.in-addr.arpa nameserver = ns02.jdc01.pa.comcast.net.
33.68.in-addr.arpa nameserver = ns01.jdc01.pa.comcast.net.

Information from www.arin.net:
Comcast Cable Communications, Inc.
JUMPSTART-1 (NET-68-32-0-0-1)
68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc.
JUMPSTART-BALTIMOR-B1 (NET-68-33-0-0-1)
68.33.0.0 - 68.34.127.255

10. Correlations from previous students practicals:

This is done along with each alert discussion.

11. Insights into certain internal machines:

A list of internal machines is given that should be further investigated for compromise or malicious activity.

MY.NET.84.151: More than 130 IP addresses seem to be connecting to the backdoor (port 65535) on MY.NET.84.151
MY.NET.6.40: SMTP server. It has generated 61 'Red worm' alerts. There is a chance that these are false alarms but being an important machine it is recommended that a check be done.
MY.NET.88.193, MY.NET.198.220:
Involved in suspicious activity on backdoor port 65535. MY.NET.198.220 is also actively scanning external hosts for port 6667
MY.NET.179.77: Suspicious activity with 218.145.25.109

```
inetnum:     218.144.0.0 - 218.159.255.255
netname:     KORNET
descr:       KOREA TELECOM
descr:       Network Management Center
```

The attacker has done an IIS Unicode attack and later the internal m/c is connecting to port 27374, which is a suspicious port. However, between these two activities there is a long duration. It is however, recommended to check the internal machine for possible compromise.

```
01/16-06:07:43.274300  [**] spp_http_decode: IIS Unicode attack detected [**]
218.145.25.109:2789 -> MY.NET.179.77:80
01/16-06:08:45.265582  [**] spp_http_decode: IIS Unicode attack detected [**]
218.145.25.109:12592 -> MY.NET.179.77:80
01/16-06:11:10.698105  [**] spp_http_decode: IIS Unicode attack detected [**]
218.145.25.109:36152 -> MY.NET.179.77:80
01/18-15:47:14.916384  [**] Possible trojan server activity [**] MY.NET.179.77:80 ->
218.145.25.109:27374
01/18-15:47:15.120522  [**] Possible trojan server activity [**] 218.145.25.109:27374 ->
MY.NET.179.77:80
01/18-15:47:15.120545  [**] Possible trojan server activity [**] 218.145.25.109:27374 ->
MY.NET.179.77:80
```

MY.NET.113.4,MY.NET.84.193, MY.NET.105.204, MY.NET.90.212, MY.NET.178.101, MY.NET.118.6, MY.NET.114.45 MY.NET.198.185, MY.NET.153.143, MY.NET.91.104,MY.NET.86.106, MY.NET.90.242, MY.NET.150.209, MY.NET.114.88, MY.NET.106.228
These machines have triggered the Wacthlist alerts (Watch list 000220 IL-ISDNNET-990517) more than 100 times. It is recommended that the machines be checked further.
130.85.190.90 (MY.NET.190.90):
This machine is scanning external IP addresses for port 139 - NetBIOS Session (TCP), Windows File and Printer Sharing

130.85.82.32 (MY.NET.82.32): Suspicious external port 80 scan

MY.NET.105.48: Seems to be using IRC. Depending on whether this is allowed by university policy, it can be allowed or disallowed.There are malicious versions of IRC clients; for e.g. ircII version 2.2.9 had a Trojan installed in it [1]. Seems to be connecting to IRC servers - 128.242.65.30, 63.98.19.244

I connected to port 6667 on these machines to verify if they actually run IRC services or whether 6667 is used as an ephemeral port. I got a response which confirms it is actually IRC servers

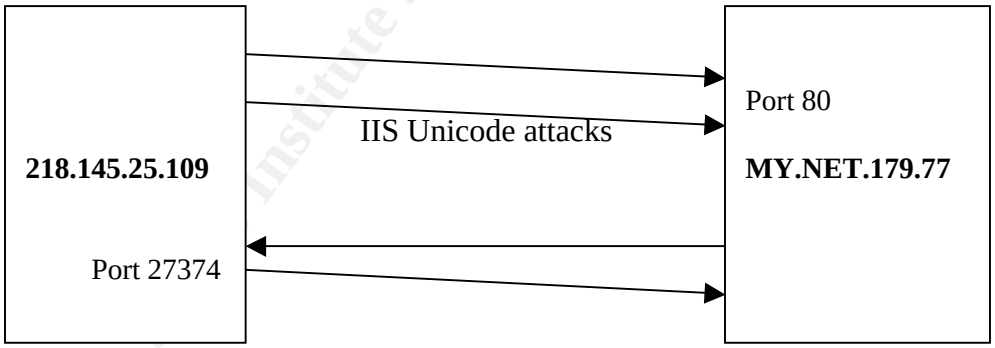
```
marks% telnet 128.242.65.30 6667
Trying 128.242.65.30...
Connected to 128.242.65.30.
Escape character is '^]'.
NOTICE AUTH :*** Looking up your hostname...
NOTICE AUTH :*** Checking Ident
NOTICE AUTH :*** Found your hostname
NOTICE AUTH :*** Got Ident response
```

Reference:

[1] http://www.cert.org/advisories/CA-1994-14.html

MY.NET.88.168: Seems to be using IRC. Depending on whether this is allowed by university policy, it can be allowed or disallowed.It was confirmed that the machine was connecting to IRC servers by the same method as mentioned above.

12. Link Graph:



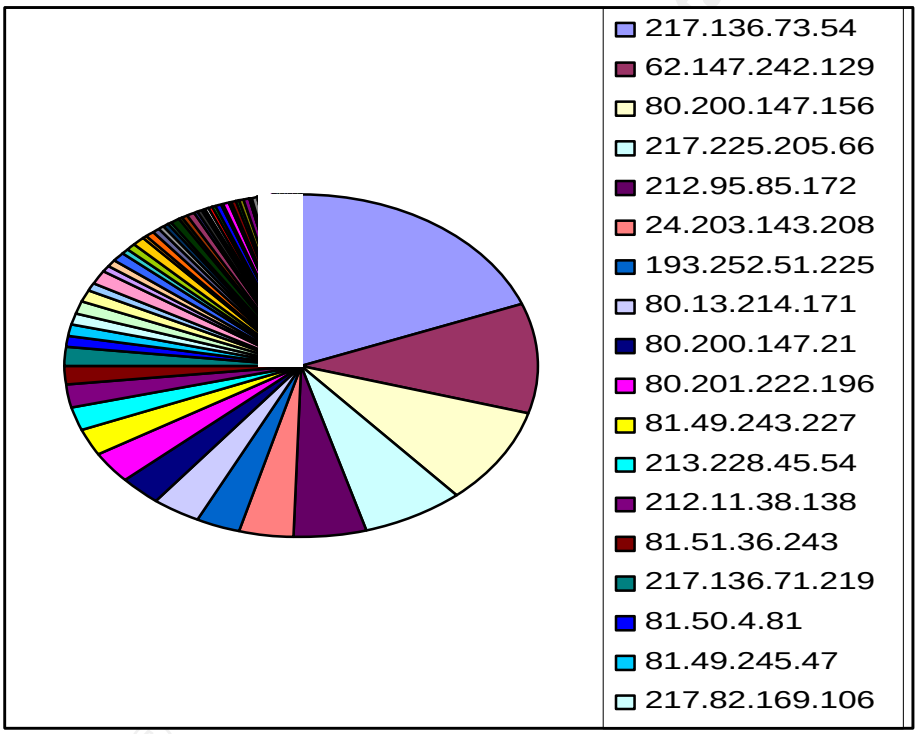
1. The diagram above shows a scenario in which an external IP, 218.145.25.109 can be seen attempting IIS Unicode attack on MY.NET.179.77. There is no information whether these attacks were successful. However, after 2 days, a suspicious connection from the web server to the external machine on port 27374 can be noted.

   The snippet of traffic is shown –

```
01/16-06:07:43.274300  [**] spp_http_decode: IIS Unicode attack
detected [**] 218.145.25.109:2789 -> MY.NET.179.77:80
01/16-06:08:45.265582  [**] spp_http_decode: IIS Unicode attack
detected [**] 218.145.25.109:12592 -> MY.NET.179.77:80
01/16-06:11:10.698105  [**] spp_http_decode: IIS Unicode attack
detected [**] 218.145.25.109:36152 -> MY.NET.179.77:80
01/18-15:47:14.916384  [**] Possible trojan server activity [**]
MY.NET.179.77:80 -> 218.145.25.109:27374
01/18-15:47:15.120522  [**] Possible trojan server activity [**]
218.145.25.109:27374 -> MY.NET.179.77:80
..
```

The port 27374 is associated with many Trojans like subseven Trojan, Lion and ramen worm. There is of port 27374 being selected as an ephemeral port for connecting to web server. But, the connection seems to start from the web server side (unless IDS is dropping packets). The web server should be checked.

2.



MY.NET.84.151 was the top talker for the 'Red worm alert' (See alert #1). Since 65535 is a valid port, I thought I would analyze the alert a bit more to check if it is a false alarm. Above 130 IP addresses were connecting to the port 65535 on MY.NET.84.151 and the distribution of traffic according to the source IP addresses is shown in the above Pie chart. (All the IP addresses are not shown on the legend). The large number of source IP addresses and the amount of traffic lessens the possibility of the false alarm.

13.Analysis process used:

All the alert files were concatenated into a single alert file and the analysis was done on that. Same procedure was done for the scans and the oos files.

The analysis of the files were done using TCL scripts as well as standard unix commands – awk, grep, uniq, sort etc.

```
#!/usr/bin/tcl
set file [lindex $argv 0]
set fp [open $file r]
set wp [open "Names.txt" w]
set wp1 [open "SrcAddr.txt" w]
set wp2 [open "DstAddr.txt" w]
set wp3 [open "SrcPort.txt" w]
set wp4 [open "DstPort.txt" w]
set data [read $fp]
set data [split $data "\n"]
foreach line $data {
        # We don't want the portscan alerts
        set ps [regexp "spp_portscan" $line]
        if {$ps == 0} {
                #Here we capture the name from the Alert-line
                set alert [ lindex [split $line "]"] 1]
                set addresses [ lindex [split $line "]"] 2]
                set alert [ lindex [split $alert "\["] 0]
                if [info exists Array($alert)] {
                        set Array($alert) [expr $Array($alert) + 1]
                } else {
                        set Array($alert) [expr 0 + 1]
                }
                #puts $addresses
                set src_address [lindex [ split [lindex [split $addresses "-"] 0] ":"] 0]
                set src_port [lindex [ split [lindex [split $addresses "-"] 0] ":"] 1]
                set addresses [lindex [ split [lindex [split $addresses "-"] 1] ">"] 1]
                set dst_address [lindex [ split $addresses ":"] 0]
                set dst_port [lindex [ split $addresses ":"] 1]
                if [info exists Src_address($src_address)] {
                        set Src_address($src_address)
                                [expr  $Src_address($src_address) + 1]
                } else {
                        set Src_address($src_address) [expr 0 + 1]
                }
                if [info exists Src_port($src_port)] {
                        set Src_port($src_port) [expr $Src_port($src_port) + 1]
                } else {
```

```
                set Src_port($src_port) [expr 0 + 1]
            }
            if [info exists Dst_address($dst_address)] {
                set Dst_address($dst_address)
                    [expr $Dst_address($dst_address) + 1]
            } else {
                set Dst_address($dst_address) [expr 0 + 1]
            }
            if [info exists Dst_port($dst_port)] {
                set Dst_port($dst_port) [expr $Dst_port($dst_port) + 1]
            } else {
                set Dst_port($dst_port) [expr 0 + 1]
            }
        }
}
set array_index [array names Array]
puts $wp "Alert names"
puts $wp "-----------"
foreach index $array_index {
        puts $wp "$Array($index) $index"
}
set array_index [array names Src_address]
foreach index $array_index {
        puts $wp1 "$Src_address($index) $index"
}
set array_index [array names Dst_address]
foreach index $array_index {
        puts $wp2 "$Dst_address($index) $index"
}
set array_index [array names Src_port]
foreach index $array_index {
        puts $wp3 "$Src_port($index) $index"
}
set array_index [array names Dst_port]
foreach index $array_index {
        puts $wp4 "$Dst_port($index) $index"
}
```

This program was used to parse the alert file. This produced alert names and their frequency, and similarly for source IP, destination IP, source port and destination port. The same program could be used on subset of alert files. For example to analyze just one type of alert, first use grep to create the subset of alert file and then followed by the above program.
A variation of the above program was used to parse the scan file too. The program is not listed due to space constraints.

Last Updated: September 23rd, 2017

Upcoming Training

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
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 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 Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
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 London February 2018	London, United Kingdom	Feb 05, 2018 - Feb 10, 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 Secure Canberra 2018	Canberra, Australia	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced