Global Information Assurance Certification Paper

GIAC Certified Intrusion Analysis (GCIA)
Practical Assignment
Version 3.4

Michael T Meacle

June 26th, 2004

Turbo Charging nIDS with Apache Reverse Proxy
"The smart (poor) man's HTTP protocol Scrubber"

TABLE of Contents

Preliminary Notes

Throughout this document there are numerous references to severity. As everyone's measure of severity will differ slightly to assist the reader I have included the definition taken directly from the assignment. [1] This idea, as has a lot of the structure of this assignment, is based on the previous GCIA submission by Sylvain Randier. [2]

1. Evaluation of Severity

Severity =
(Criticality + Lethality) – (System countermeasures + Network countermeasures)

Where:

Measure	Description (extracted from assignment 1)
Criticality	Is a measure of how critical the target system is.
Lethality	Is a measure of how severe the damage to the target would be if the attack succeeded.
System Countermeasure	Is a measure of the strength of the defensive mechanisms in place on the host itself.
Network Countermeasure	Is a measure of the strength of the defensive mechanisms in place on the network.

Each component should be allocated a value of between 1 (lowest) and 5 (highest).

The more positive the calculated severity is the more likely the event will have negative impact on your organisation.

A consistent method used to calculate the severity of each event, thus identifying events of interests, can be invaluable as a method of triage during a large-scale attack. [3]

Question 1: Turbo Charging IDS with Apache Reverse Proxy

Option 1 - IDS Technology or Challenge

1 Abstract

Network Intrusion Detection System (nIDS), such as snort, can provide an effective and timely alerting to malicious activity. Unfortunately due to a brilliant article by Ptacek [4] some IDS can be rendered totally ineffective in detecting some attacks. In this paper I look at a cheap solution for a small organisation with a small online presence to improve the effectiveness of their nIDS.

I will demonstrate in a lab how to use a reverse proxy to sanitise all http traffic both at the network and transport layer. As a result of removing any ambiguities in traffic flows seen both by the IDS and ultimately any web server, any nIDS can be made to operate more effectively.

However getting the 'simple' lab to work proved much more challenging than expected. While conceptually it is a very simple lab; once one starts analysis of alerts, capture sizes and individual frames it becomes apparent that more refinement was required. In fact, I have included 4 distinct phases in the evolution of the lab. This evolution to the successful lab has been included to assist understanding and facilitate replication by fellow peers.

Relative comparison of the results obtained in the final lab will be used to demonstrate how effective a simple inline proxy can be in improving the performance of any nIDS.

The results speak for themselves, without a doubt an inline reverse proxy can be a very cost effective way to improve the performance of nIDS monitoring of http traffic.

2 Introduction

Many years ago I read, and re-read the evolutionary article "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" [4]. At the time I was actively involved with System administration, Network performance and firewall maintenance. Surprisingly while I had knowledge of IDS's my primary reason for reading the article was not to understand NIDS but to get a better understanding of TCP/IP for use in my day-to-day job. Ptacek's article outlined a number of techniques to attack a network element oblivious to a number of commonly available commercial NIDS.

Around the same time I read an interesting article "Know Your Enemy: Statistics". [5] One interesting point outlined in the article by the Honeynet Project is the ability of a honeynet to help minimise false positive's and false negatives as 'all' data is of interest.

A couple years ago in a flash of brilliance I thought why not reduce false positives and negatives by using some device to remove all traces of the tactics outlined by Ptacek. I subsequently bounced, off a number of work colleagues, the idea of using

a reverse proxy as a cost effective way to improve IDS effectiveness. Due to a lack of time I put the idea away for a rainy day.

Fast-forward to January 2004 and I had the fortune to attend a SANS GIAC IDS course in Sydney. Now the challenging part …. "THE PRACTICAL".

3 Background

As part of my practical I decided to investigate "my" idea of reducing false positives and negatives. At least I thought it was my idea until I consulted google [6]. I soon learnt that I wasn't alone in my endeavour to turbo charge a nIDS by improving the quality of data an nIDS has to work with. In fact, not only has it been thoroughly researched, due to research timings, it has two different names normalisation [7] and scrubbing [8].

I also identified two relevant practicals on SANS reading room. The first by Ian Martin gives a very detailed chronological analysis and culminated in an in-depth look at normalisation. [9] The second article by Benjamin Sapiro, as part of his GSEC practical, focused on both commercial and free packet scrubbers. [10]

My intention is to demonstrate how a small business with a small online presence can use apache in reverse proxy mode to scrub/normalise http prior to being monitored by a suitable nIDS. I have intentionally used free, commonly available software so that this is both achievable and affordable to small organisations or the budding student wishing to experiment.

4 Lab

Now for the easy part, or at least I thought so, to demonstrate how apache's reverse proxy can be used in increasing the effectiveness of a NIDS.

Where effectiveness means:

- Less false positives
- Less false negatives
- Less noisy data for an analyst to analyse

To do this I will run two sets of tests. The first was http vulnerability scan using Nessus [11]. Nessus 2.0.10 for Linux was used. In this test I enabled all the "CGI abuses" plugins. In the second test I used wget [12] to make a number of valid http page requests; 6 small gifs totalling 24k. For this second test the standard version of wget provided with Fedora was used, rpm reported wget-1.8.2-15.3.

During each test I would use a suitable tool to modify the TCP/IP data stream as outlined in Ptacek's brilliant article [4]. While a number of tools could be used, two tools, both by the Dug Song, were identified as they provided the easy ability to modify the packets. The first called fragroute [13] can only be used to manipulate traffic originating from the host it is running on. The second tool fragrouter [14] can only work on traffic passing through the host it is operating on.

Fragroute 1.2-1 and Fragrouter 1.6-0 were both obtained as rpm's from Dag Wieers Apt Repository [15]

I subsequently re-ran each test with apache reverse proxy inline. A full binary capture of each test was saved using tcpdump [16]. Each capture was then analysed with snort [17]. The version of snort used was 2.1.2 (Build 25). Relative comparisons were then made for each of the tests against each baseline test.

While it could be argued that my testing methodology is flawed due to the fact I never mixed valid traffic with invalid traffic as detailed in the article "Intrusion Detection Testing and Benchmarking Methodologies"[18]. I would argue that I am doing subjective evaluation based on relative performance as opposed to absolute performance.

All tests were performed on decommissioned hardware. Three boxes (mangler, gateway, www) were old Ipex Centra with quad Zeon 550MHz processors and 1GB ram. The client pc was AMD Athlon 1.15GHz with 512MB ram. All were fresh builds based on a basic custom install of Fedora FC1.

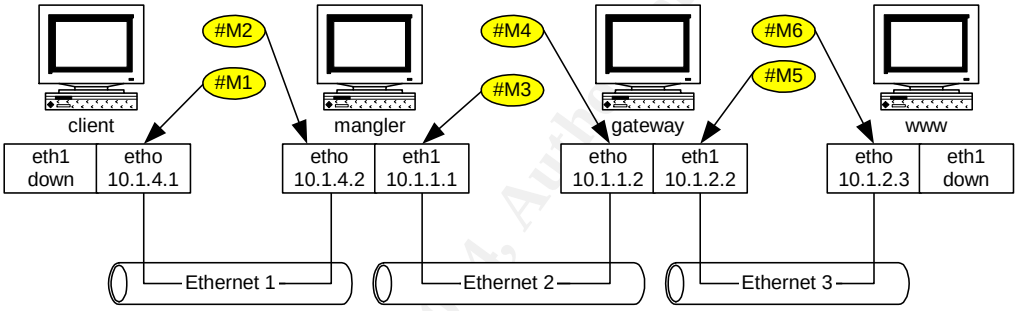


Figure 1: Basic Lab Layout

Common components of lab for all 4 attempts was as follows:

- www
  - standard install of apache configured to serve up pages
- gateway (e.g server running the reverse proxy)
  - no iptables and standard ip forwarding enabled
  - apache configured to listen on 10.1.1.2 and reverse proxy all requests to 10.1.2.3, see differential config in figure 2.
  - binary capture files were taken on both eth0 and eth1
- client
  - snort [17] for analysis of captures

```
[root@gateway root]$ diff httpd.conf httpd.conf.orig
259d258
< ServerName 10.1.1.2:80
290d288
<
1053,1058d1050
<
<
< <VirtualHost 10.1.1.2:80>
<     ProxyPass / http://10.1.2.3/
<     ProxyPassReverse / http://10.1.2.3/
```

```
< </VirtualHost>
```

Figure 2: Difference between an actual Apache configuration and original configuration file on the gateway server.

2.1 Attempt 1: Fragroute (Fail)

This was my initial attempt at demonstrating and measuring the effectiveness of apache reverse proxy.

The lab was set as follows:

- See previous common configurations
- mangler
  - nessus[11]
  - wget[12]
  - fragroute [13]
- client was not used; all tests were initiated from mangler

While wget was able to transfer files without difficulty, nessus refused to work. As shown in the following analysis of a capture on gateway interface eth0 (#m4) nessus could not perform any tests as tcp resets were being generated as soon as syn/ack were received.

```
D:\sans\T3ids\data>windump -n -r test1 "port 80"
17:56:02.396614 IP 10.1.1.1.24273 > 10.1.2.3.80: S
4268841998:4268841998(0) win 8

17:56:02.397360 IP 10.1.2.3.80 > 10.1.1.1.24273: S
505242930:505242930(0) ack 42 68841999 win 5840 <mss 1460> (DF)

17:56:02.397604 IP 10.1.1.1.24273 > 10.1.2.3.80: R
4268841999:4268841999(0) win 0 (DF)

The yellow highlight being the most important.
```

Footnote: Challenge for the astute reader – determine if it is possible to get this to work using iptables as outlined in 2.4

2.2 Attempt 2: Fragrouter without IPTables (Fail)

Time for a different tack; try to use fragrouter[14] as opposed to fragroute[13]. The lab was set as follows:

- See previous common configurations
- mangler
  - fragrouter [14]
- client
  - nessus [11]
  - wget [12]

All tests using both wget and nessus appeared to complete successfully. However I was concerned why the wget reverse proxied baseline tests had approximately twice as many packets from client to gateway (3553) than from gateway to www (1629). For this baseline test one would expect the same amount of packets for client to www irrespective of the inline proxy.

After close inspection of the wget baseline captures between gateway and www I was able to identify each packet from client to www was duplicated. In fact I was able to identify that the original packet was passed straight through while the second packet, some 28usec latter, appeared to originate from the fragrouter software. This was determined by the fact that both (client to www) packets appeared identical except the second packet ttl was 64 as opposed to the original packet with a ttl of 63.

```
Original Packet
18:07:54.860174 IP (tos 0x10, ttl 63, id 4773, len 59) 10.1.3.1.32775 >
10.1.2.3.80: P [tcp sum ok] 1:8(7) ack 1 win 5840 <nop,nop,timestamp
582635 269265558> (DF)

Fragrouter Generated Packet
18:07:54.860202 IP (tos 0x10, ttl 64, id 4773, len 59) 10.1.3.1.32775 >
10.1.2.3.80: P [tcp sum ok] 1:8(7) ack 1 win 5840 <nop,nop,timestamp
582635 269265558> (DF)

Each of the highlights shows important relationships between the two
traces.
The yellow highlight being the most important.
```

2.3 Attempt 3: Fragrouter with default Fedora IPTables (Fail)

In the previous attempt I identified that the original packet from client to www was duplicated. I decided to use IPTables on mangler to drop the original packet from client to www. This was easily implemented as during the initial install I installed iptables with a default restrictive policy. I had to manually start the firewall software as I had previously stopped it.

The rule I added to drop the original packet from client to www was entered as follows using the standard iptables CLI.

```
root@mangler# iptables –I FORWARD –p tcp –dport 80 –j DROP
```

The rest of the lab was set as follows:

- See previous common configurations
- mangler
  - fragrouter [14]
- client
  - nessus [11]
  - wget [12]

Once again all tests using both wget and nessus appeared to complete successfully. But on closer inspection, I became concerned why the number of packets in the baseline wget non-proxied request (1651) was almost identical to the number of packets in fragment test [-F1] (1654). Considering fragrouter was reporting fragmentation was taking place the number of packets in the capture indicated otherwise.

```
Extract Showing Fragrouter Generating 3 Fragments
[root@mangler sbin]# ./fragrouter -p -i eth0 -F1
fragrouter: frag-1: ordered 8-byte IP fragments
10.1.3.1.32777 > 10.1.2.3.80: P ack 1644734489 win 5840
```

```
<nop,nop,timestamp 621358 269303928> (frag 4201:32@0+) [tos 0x10]
10.1.3.1 > 10.1.2.3: (frag 4201:8@32+) [tos 0x10]
10.1.3.1 > 10.1.2.3: (frag 4201:8@40) [tos 0x10]

Extract Showing The Defragmented Packet captures at #M4
D:\sans\T3gcia\data>windump -n -r test3 -S -v

18:14:22.049776 IP (tos 0x10, ttl 64, id 4201, len 68)
  10.1.3.1.32777 > 10.1.2.3.80: P [tcp sum ok]
  2193254641:2193254657(16) ack 1644734489 win 5840
  <nop,nop,timestamp 621358 269303928>

Each of the highlights shows important relationships between the two
traces.
The yellow highlight being the most important.
```

I stopped the firewall software and reran a simple fragmented wget. Analysis of the packets on gateway indicated effective ip fragmentation. So it appeared iptables was defragmenting fragrouters' fragmented ip packets prior to them leaving the mangler server. Contained within Rusty Russell's packet filtering HowTo is a single throwaway line stating connection tracking, if enabled, will defragment all packets. [19].

A quick check of loaded modules when iptables was running, confirmed that the iptables connection-tracking module was loaded.

```
Extract of loaded modules showing
[root@mangler sbin]# lsmod
Module                  Size  Used by    Not tainted
<cut>
ipt_REJECT               4344   1  (autoclean)
ipt_state                1080   3  (autoclean)
ip_conntrack            28840   1  (autoclean) [ipt_state]
iptable_filter           2444   1  (autoclean)
ip_tables               15264   3  [ipt_REJECT ipt_state
iptable_filter]
<cut>

The yellow highlight being the most important.
```

2.4 Attempt 4: Fragrouter with Custom IPTables (Success)

In the previous attempt I identified that fragmentation was not working as expected due to the fact iptables connection tracking module was being loaded when iptables was being started. I initially flushed all tables, which contained connection-tracking directives hoping, but not expecting, iptables to automatically unload the module. As expected it remained loaded.

To allow iptables to start without any reference to connection tracking I edited to default firewall rules and removed all references to connection tracking.

```
Stripped IPTABLES startup configuration
[root@mangler /]# cat /etc/sysconfig/iptables
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
```

```
:OUTPUT ACCEPT [0:0]
-A FORWARD -p tcp --dport 80 -j DROP
COMMIT

The yellow highlight being the most important.
```

The rest of the lab was set as follows:

- See previous common configurations
- mangler
  - fragrouter [14]
  - iptable rule to drop original packets from client
- client
  - nessus [11]
  - wget [12]

All tests using both wget and nessus now appear to complete successfully; what's more initial analysis indicated tests were valid.

5 Results

A lot of data was collected during these tests. Each of the final 36 capture files kept and analysed were captured on the gateway server [#M4, #M5]. The total size of the capture files was 950MB.

A script was written to analyse each capture twice with snort. All snort signatures and all pre-processors were enabled for each analysis. Each run saved its result in a separate directory for further analysis. In the first run the stream4 pre-processor had evasion alerting disabled; snort's default when stream4 pre-processor is enabled. In the second run the stream4 pre-processor had evasion alerting enabled.

To assist in understanding the name of each capture the following naming convention was used:

Fragment	**Comment**
nessus	A test in which nessus was used
wget	A test in which wget was used
np	No proxy e.g. directly connecting to www
px	Proxy, e.g. connecting via reverse proxy
eth0	Eth0 on gateway
eth1	Eth1 on gateway
	Fragrouter test
B1	-B1: base-1: normal IP forwarding
F1	-F1: frag-1: ordered 8-byte IP fragments
F7	-F7: frag-7: ordered 16-byte fragments, fwd-overwriting
T7	-T7: tcp-7: 3-whs, ordered 1-byte segments, interleaved null segments
C2	-C2: tcbc-2: 3-whs, ordered 1-byte segments, interleaved SYNs
M1	-M1: misc-1: Windows NT 4 SP2 - http://www.dataprotect.com/ntfrag/

tcpdump capture		Time Info				Snort
Filename	Size	real	user	sys	Total	# Alerts
nessus-np-eth0-B1	5077071	1.21	0.95	0.1	2.26	410
nessus-np-eth0-F1	10320228	1.77	1.34	0.13	3.24	414
nessus-np-eth0-F7	16598843	2.11	1.52	0.4	4.03	3974
nessus-np-eth0-T7	246225452	21.28	18.35	1.43	41.06	4528
nessus-np-eth1-B1	5074085	1.33	0.86	0.18	2.37	410
nessus-np-eth1-F1	8624078	1.59	1.33	0.1	3.02	414
nessus-np-eth1-F7	14472241	2.19	1.41	0.5	4.1	3974
nessus-np-eth1-T7	243315886	21.62	18.61	1.36	41.59	4536
nessus-px-eth0-B1	5042589	1.39	0.98	0.11	2.48	418
nessus-px-eth0-F1	13598452	2.44	1.54	0.25	4.23	396
nessus-px-eth0-F7	13783612	1.86	1.32	0.27	3.45	3093
nessus-px-eth0-T7	351555232	29.91	25.63	1.73	57.27	3748
nessus-px-eth1-B1	5021455	1.35	0.96	0.07	2.38	0
nessus-px-eth1-F1	4966478	1.15	0.92	0.09	2.16	0
nessus-px-eth1-F7	4979965	1.19	0.9	0.1	2.19	0
nessus-px-eth1-T7	719731	0.73	0.54	0.08	1.35	0

Table 1: Nessus capture with snort's Evasion Alert Disabled

tcpdump capture		Time Info				Snort
Filename	Size	real	user	sys	Total	# Alerts
nessus-np-eth0-B1	5077071	1.73	0.97	0.12	2.82	524
nessus-np-eth0-F1	10320228	1.89	1.26	0.19	3.34	712
nessus-np-eth0-F7	16598843	2.26	1.58	0.4	4.24	3980
nessus-np-eth0-T7	246225452	72.91	30.52	35.3	138.73	576232
nessus-np-eth1-B1	5074085	1.46	0.97	0.1	2.53	524
nessus-np-eth1-F1	8624078	1.5	1.26	0.2	2.96	712
nessus-np-eth1-F7	14472241	2.2	1.51	0.42	4.13	3980
nessus-np-eth1-T7	243315886	70.61	30.12	33.34	134.07	576224
nessus-px-eth0-B1	5042589	1.27	0.91	0.13	2.31	420
nessus-px-eth0-F1	13598452	1.76	1.46	0.22	3.44	402
nessus-px-eth0-F7	13783612	1.85	1.24	0.35	3.44	3101
nessus-px-eth0-T7	351555232	83.88	39.06	34.06	157	707134
nessus-px-eth1-B1	5021455	1.57	0.96	0.09	2.62	172
nessus-px-eth1-F1	4966478	1.22	0.92	0.11	2.25	48
nessus-px-eth1-F7	4979965	1.31	0.92	0.08	2.31	68
nessus-px-eth1-T7	719731	0.88	0.55	0.07	1.5	4

Table 2: Nessus capture with snort's Evasion Alert Enabled

tcpdump capture		Time Info				Snort
Filename	Size	real	user	sys	Total	# Alerts
wget-np-eth0-B1	7868	1.09	0.51	0.05	1.65	0
wget-np-eth0-C2	2015202	0.81	0.6	0.08	1.49	0
wget-np-eth0-F1	12680	0.7	0.5	0.05	1.25	0
wget-np-eth0-F7	13282	0.72	0.52	0.03	1.27	5
wget-np-eth0-M1	7920	0.73	0.47	0.08	1.28	0
wget-np-eth1-B1	7868	0.74	0.46	0.08	1.28	0
wget-np-eth1-C2	1978322	0.87	0.54	0.13	1.54	0
wget-np-eth1-F1	11322	0.69	0.47	0.08	1.24	0

tcpdump capture		Time Info				Snort
Filename	Size	real	user	sys	Total	# Alerts
wget-np-eth1-F7	12038	0.75	0.46	0.08	1.29	5
wget-np-eth1-M1	7786	0.7	0.45	0.09	1.24	0
wget-px-eth0-B1	7570	0.71	0.5	0.04	1.25	0
wget-px-eth0-C2	1260389	0.85	0.55	0.08	1.48	0
wget-px-eth0-F1	12926	0.67	0.47	0.08	1.22	0
wget-px-eth0-F7	13148	0.73	0.48	0.06	1.27	5
wget-px-eth0-M1	7570	0.72	0.5	0.04	1.26	0
wget-px-eth1-B1	7754	0.74	0.46	0.08	1.28	0
wget-px-eth1-C2	7754	0.81	0.48	0.08	1.37	0
wget-px-eth1-F1	7754	0.68	0.47	0.07	1.22	0
wget-px-eth1-F7	7754	0.72	0.47	0.07	1.26	0
wget-px-eth1-M1	7754	0.73	0.49	0.06	1.28	0

Table 3: wget capture with snort's Evasion Alert Disabled

tcpdump capture		Time Info				Snort
Name	Size	real	user	sys	Total	# Alerts
wget-np-eth0-B1	7868	0.78	0.47	0.07	1.32	0
wget-np-eth0-C2	2015202	1.46	0.81	0.48	2.75	9864
wget-np-eth0-F1	12680	0.76	0.45	0.08	1.29	0
wget-np-eth0-F7	13282	0.73	0.49	0.03	1.25	5
wget-np-eth0-M1	7920	0.76	0.44	0.11	1.31	0
wget-np-eth1-B1	7868	0.74	0.41	0.11	1.26	0
wget-np-eth1-C2	1978322	1.77	0.8	0.66	3.23	10094
wget-np-eth1-F1	11322	0.7	0.43	0.12	1.25	0
wget-np-eth1-F7	12038	0.99	0.48	0.08	1.55	5
wget-np-eth1-M1	7786	0.7	0.45	0.1	1.25	0
wget-px-eth0-B1	7570	0.74	0.47	0.08	1.29	0
wget-px-eth0-C2	1260389	1.13	0.62	0.28	2.03	4574
wget-px-eth0-F1	12926	0.71	0.42	0.13	1.26	0
wget-px-eth0-F7	13148	1.02	0.46	0.07	1.55	5
wget-px-eth0-M1	7570	0.7	0.44	0.12	1.26	0
wget-px-eth1-B1	7754	0.72	0.41	0.13	1.26	0
wget-px-eth1-C2	7754	0.8	0.45	0.1	1.35	0
wget-px-eth1-F1	7754	0.7	0.46	0.07	1.23	0
wget-px-eth1-F7	7754	0.81	0.44	0.1	1.35	0
wget-px-eth1-M1	7754	0.75	0.4	0.13	1.28	0

Table 4: wget capture with snort's Evasion Alert Enabled

Tables 1 to 4 shows a summary of the raw analysis of each capture. They contain a lot of data and are shown for completeness in obtaining table 5.

	Disable Evasion Alert				Enable Evasion Alert			
Gateway eth1	No Proxy		Proxy		No Proxy		Proxy	
Test	Total	# Alerts	Total	# Alerts	Total	# Alerts	Total	# Alerts
nessus B1	2.37	410	2.38	0	2.53	524	2.62	172
nessus F1	3.02	414	2.16	0	2.96	712	2.25	48
nessus F7	4.1	3974	2.19	0	4.13	3980	2.31	68

	Disable Evasion Alert				Enable Evasion Alert			
Gateway eth1	**No Proxy**		**Proxy**		**No Proxy**		**Proxy**	
Test	**Total**	**# Alerts**	**Total**	**# Alerts**	**Total**	**# Alerts**	**Total**	**# Alerts**
nessus T7	41.59	4536	1.35	0	134.07	576224	1.5	4
wget B1	1.28	0	1.28	0	1.26	0	1.26	0
wget C2	1.54	0	1.37	0	3.23	10094	1.35	0
wget F1	1.24	0	1.22	0	1.25	0	1.23	0
wget F7	1.29	5	1.26	0	1.55	5	1.35	0
wget M1	1.24	0	1.28	0	1.25	0	1.28	0
		9339		**0**		**591539**		**292**

Table 5: Summary of Analysis for Comparison

Table 5 is a summary of the most important fields from tables 1-4. To ensure table 5 is as concise as possible it only contains output of the gateway server (#M5). Data is displayed both when used as a reverse proxy and as a normal forwarding router.

The initial review of the values contained in table 5 raises some interesting results. The most glaring is variation (0 – 591 539) in the number of alerts that are created for the same traffic. Table 5 compares and contrast between using an inline proxy and not having the inline proxy to normalise the traffic. Additionally table 5 shows another dimension that is whether enabling the evasion alert option to the stream4 pre-processor has any effect.

Depending on how the same capture files are analysed snort will produce either 0, 292, 9339 or 591 539 alerts.

Which one is most correct?

Which one would your prefer to manage?

In trying to answer this, let us start with the wget results. Each test involved downloading 5 small gif's, total size 24kB. Yet 5.4MB of data was transferred. Depending on how snort was configured the analyst may have 0,5 or 10099 alerts to wade through. Remember these were simple obfuscated successful wget's with no malicious traffic. So in effect they are all false positives. All these false positives put load on both the analyst and the IDS infrastructure. In fact if the analyst regularly had to look through 11k of false positives he would either miss the real events or stop looking.[20]

An astute reader might have noticed in the proxied nessus scan when the anti-evasion was enabled that some 292 alerts were created. These, especially the 172 in the baseline concerned me. On closer inspection I discovered they were all categorised as "TCP CHECKSUM CHANGED ON RETRANSMISSION (possible fragroute) detection". Considering we had a proxy inline to remove all ambiguities this seemed odd. On closer inspection of alerts in B1 and F1 captures I was able to confirm each time it happened the proxy had retransmitted a tcp packet approximately 210msec after the first. In all cases the two packets were identical except for the first tcp timestamps had changed. This new timestamp results in a new tcp checksum. Unfortunately snort stream4 pre-processor expects the same checksum for retransmitted tcp frames.

```
Extract from nessus-px-eth1-B1 alert
[**] [111:16:1] (spp_stream4) TCP CHECKSUM CHANGED ON RETRANSMISSION
(possible fragroute) detection [**]
05/08-12:09:04.590814 10.1.2.2:60612 -> 10.1.2.3:80
TCP TTL:64 TOS:0x0 ID:11869 IpLen:20 DgmLen:460 DF
***AP*** Seq: 0xA94DCD06  Ack: 0xC2D9CC70  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 176022388 210223287

Extract Showing The Deframented Packet
[root@gateway /]# tcpdump -n -s 0 -r nessus-px-eth1-B1 'src port 60612
and tcp[tcpflags] & tcp-push != 0' -X

12:09:04.383897 10.1.2.2.60612 > 10.1.2.3.http: P
  2840448262:2840448670(408) ack 3269053552 win 5840
  <nop,nop,timestamp 176022367 210223287> (DF)
0x0000   4500 01cc 2e5c 4000 4006 f2c9 0a01 0202  E....\@.@.......
0x0010   0a01 0203 ecc4 0050 a94d cd06 c2d9 cc70  .......P.M.....p
0x0020   8018 16d0 39ad 0000 0101 080a 0a7d e35f  ....9........}._
0x0030   0c87 c0b7 4745 5420 2f50 5355 7365 722f  ....GET./PSUser/
<cut>

12:09:04.590814 10.1.2.2.60612 > 10.1.2.3.http: P
  0:408(408) ack 1 win 5840
  <nop,nop,timestamp 176022388 210223287> (DF)
0x0000   4500 01cc 2e5d 4000 4006 f2c8 0a01 0202  E....]@.@.......
0x0010   0a01 0203 ecc4 0050 a94d cd06 c2d9 cc70  .......P.M.....p
0x0020   8018 16d0 3998 0000 0101 080a 0a7d e374  ....9........}.t
0x0030   0c87 c0b7 4745 5420 2f50 5355 7365 722f  ....GET./PSUser/
<cut>
0x01c0   2e6c 6162 2e63 6f6d 0d0a 0d0a            .lab.com....

Each of the highlights show important relationships between the two
traces.
The yellow and red highlight being the most important.
```

While 210mSec seems excessively low for fast retransmit; RTO as low as 200mSec are now commonly seen since Linux kernel 2.4 [21].

A review of the stream4 pre-processor source code confirms for retransmitted packets of the same length only tcp checksum's are used to validate if data has been changed. The challenge here for the snort developer is to correctly handle "TCP Stream Reassembly" ambiguities as detailed in section 5.4 of Ptacek paper [4]. From my initial analysis of these false positives, I believe that this part of the snort code could be enhanced to consider the case where the only changed data in the retransmitted packet is the monotonically increasing TCP timestamp options. While this could help reduce some false positives in the case of TCP retransmits it becomes quite complex if the developer wishes to allow for repacketization [22]. If efficient checksum recalculation techniques as outlined in rfc1141[23] are used, it should be possible to substantially reduce false positives without a performance impact on Snort.

```
Extract source snort 2.1.2(25) src/preprocessors/spp_stream4.c
3735  /* check for retransmissions */
3736  returned = (StreamPacketData *) ubi_sptFind(&s->data,
(ubi_btItemPtr)spd);
3737
3738  if(returned != NULL)
3739  {
```

```
3740     DEBUG_WRAP(DebugMessage(DEBUG_STREAM, "WARNING: returned
packet         not null\n")););
3741     if(returned->payload_size == p->dsize)
3742     {
3743       /* check to see if the data has been ack'd */
3744       if(s->last_ack < pkt_seq + p->dsize)
3745       {
<cut>
3767         DEBUG_WRAP(DebugMessage(DEBUG_STREAM,
3768                    "Checking Packet Contents versus Packet
Store\n")););
3769
3770         if(returned->cksum != p->tcph->th_sum)
3771         {
3772           DEBUG_WRAP(DebugMessage(DEBUG_STREAM, "TCP Checksums not
equal\n")););
<cut>
3783             SetEvent(&event, GENERATOR_SPP_STREAM4,
3784                      STREAM4_EVASIVE_RETRANS_DATA, 1, 0, 5, 0);
3785
3786             CallAlertFuncs(p, STREAM4_EVASIVE_RETRANS_DATA_STR,
3787                                       NULL, &event);
3788

The yellow and red highlight being the most important.
```

6 Additional Benefits, Negatives and Alternatives

Thus far we have only looked at how effective an inline reverse proxy can be at reducing false positives and false negatives in http traffic. This alone can make any installation of an NIDS so much more effective however there are many more additional benefits from installing a fully configured reverse proxy:

- Centralised logging of all incoming requests
- Centralised URL and content filtering and rewriting [24]
- Load Sharing
- Centralised Authentication effectively providing single http sign-on
- Static pages can be cached taking load off back-end dynamic servers

So are reverse proxies the perfect solution to turbo charging nIDS?
Unfortunately there are some negatives:

- Latency of a very heavily loaded reverse proxy
- Risks becoming an open Internet based anonymiser if set up incorrectly e.g. see the “ProxyRequests” directive [25].
- Reliance on the security of the OS and reverse proxy application[26]
- In the case of SSL, requires certificates to be situated in DMZ[26]

With so many benefits, and research papers back to 1999/2000, there must be alternative scrubbing and normalisation solutions available. Fortunately there is, unfortunately most solutions are commercial and as such cost real money. Each solution provides scrubbing and normalisation to different levels and performance. Here is a small sample of available solutions / product feature:

- Application based firewalls; various vendors
- Intrusion Detection Prevention; various vendors
- eGap Application Firewall [26]

- Top Layer IPS 5500 [27]
- Juniper's Deep Packet Inspection [28]
- Hogwash [29]

7 Conclusion

The use of an inline reverse proxy can be very effective in improving the performance of an IDS monitoring http traffic. One of the reasons most nIDS suffer from false positives or negatives is due to ambiguities in implementations of TCP/IP stacks by various vendors. By placing an inline reverse proxy I have demonstrated a cheap and effective way of reducing the quantity and improving the quality of data an IDS has to monitor.

8 References:

[1] GIAC Certified Intrusion Analyst (GCIA),
URL: http://www.giac.org/GCIA_assignment_print.php (30/01/2004)

[2] Randier Sylvain, GCIA Practical Assignment.
URL: http://www.giac.org/practical/GCIA/Sylvain_Randier_GCIA.pdf (01/04/2004)

[3] SANS Institute,
IDS Signatures and Analysis – Part1 & 2, Chapter 4 (Sydney 2004)

[4] Ptacek, Thomas H and Newsham, Timothy N.
Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection
URL: http://www.snort.org/docs/idspaper/ (18/04/2003)

[5] Honeynet Project.
Know Your Enemy: Statistics.
URL: http://project.honeynet.org/papers/stats/ (26/11/2001)

[6] Google, URL: http://www.google.com (various)

[7] Handley, Mark and Paxson, Vern and Kreibich, Christian.
Network Intrusion Detection: Evasion, Traffic Normalization, End-To-End Protocol Semantics
URL: http://www.icir.org/vern/papers/norm-usenix-sec-01.pdf (29/04/2004)

[8] Malan, G. Robert and Watson, David and Jahanian, Farnam and Howell, Paul.
Transport and Application Protocol Scrubbing
URL: http://www.cs.ucsd.edu/~savage/cse291/papers/Malan00.pdf (15/02/2004)

[9] Martin, Ian.
Packet Level Normalisation
URL: http://www.sans.org/rr/papers/70/1128.pdf (17/02/2004)

[10] Sapiro, Benjamin.
Application Level Content Scrubbers. August 22, 2001.
URL: http://www.sans.org/rr/paper.php?id=800 (5/4/2204)

[11] Nessus,
URL: http://www.nessus.org (1/2/2004)

[12] Wget,
URL: http://www.gnu.org/software/wget/wget.html (3/4/2004)

[13] Song, Dug
Fragroute
URL: http://monkey.org/~dugsong/fragroute/ (3/4/2004)

[14] Song, Dug
Fragouter
URL: http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/fragrouter.html (3/4/2004)

[15] Wieers, Dag
DAG APT Repository
URL: http://dag.wieers.com/packages/ (3/4/2004)

[16] TCPDUMP
URL: http://www.tcpdump.org (3/4/2004)

[17] Snort,
URL: http://www.snort.org/ (1/4/2004)

[18] Athanasiades, Nicholas and Abler, Randal and Levine, John and Owen, Henry and Riley George.
Intrusion Detection Testing and Benchmarking Methodologies
URL:
http://users.ece.gatech.edu/~owen/Research/Conference%20Publications%20/sim_IIAW2003.pdf
(7/4/2004)

[19] Rusty, Russell
Linux 2.4 Packet Filtering HOWTO
URL: http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html#ss7.3
(21/4/2004)

[20] Newman, David and Snyder, Joel and Thayer, Rodney
Crying wolf: False alarms hide attacks
URL: http://www.nwfusion.com/techinsider/2002/0624security1.html (13/05/2004)

[21] Sarolahti, Pasi and Kuznetsov, Alexey
Congestion Control in Linux TCP.
URL: http://www.cs.helsinki.fi/research/iwtcp/papers/linuxtcp.pdf (15/05/2004)

[22] Stevens, W Richard
TCP/IP Illustrated Volume 1, The Protocols
Reading: Addison Wesley Professional Computing Series, 1994, PP 272,349,350

[23] Mallory, T and Kullberg, A
Incremental Updating of the Internet Checksum (January 1990)
URL: http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1141.html (26/06/2004)

[24] Apache HTTP Server 2.0, Apache Module mod_rewrite
URL: http://httpd.apache.org/doc-2.0/mod/mod_rewrite.html (23/04/2003)

[25] Apache HTTP Server Version 2.0, Apache Module mod_proxy
URL: http://httpd.apache.org/doc-2.0/mod/mod_proxy.html (23/04/2003)

[26] E-GAP Application Firewall Appliance,
A Technical Overview, January 2003
URL: http://www.whalecommunications.com/site/Whale/Corporate/Whale.asppi=30 (5/04/2004)
[requires email registration]

[27] Lindstrom, Pete
Intrusion Prevention Systems(IPS): Next Generation Firewalls
URL: http://www.toplayer.com/pdf/Whitepapers/Spire%20-%20Top%20Layer%20WP.pdf
(13/04/2004) [requires email login]

[28] Sorensen, Sarah
The Need for Pervasive Application-Level Attack Protection
URL:
http://www.netscreen.com/auth/login.jsp?_returnurl=http%3A%2F%2Fwww.juniper.net%2Fsolutions%2Fliterature%2Fwhite_papers%2Fpervasive_application_level_wp.pdf&_id=www.whitepapers
(13/05/2004) [online registration required]

[29] Haile, Jed and Larsen, Jason
Securing an Unpatchable webserver …Hogwash, last updated July 31,2001
URL: http://www.securityfocus.com/infocus/1208 (23/05/2004)

Question 2: Network Detects

1. Detect #1 .htaccess access

```
Extract from /tmp/alert
[**] [1:1129:4] WEB-MISC .htaccess access [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/27-10:45:29.116507 210.186.62.136:1361 -> 32.245.166.119:80
TCP TTL:108 TOS:0x0 ID:29485 IpLen:20 DgmLen:529 DF
***AP*** Seq: 0xC4BF5  Ack: 0xBA2B05FE  Win: 0x2180  TcpLen: 20
```

1.1 Source Of Trace:

The raw log file was obtained from http://www.incidents.org/logs/Raw/2002.9.27. As numerous students have previously pointed out before, the logfile timestamp and records contained within don't match. In fact according to tcpdump the actual period is for 27/10/2002.

```
Determining range of records in the capture
[root@snort captures]# tcpdump -n -r 2002.9.27 -tttt
10/27/2002 00:05:29.746507 255.255.255.255.31337 > 32.245.78.8.printer:
R 0:3(3) ack 0 win 0
<cut>
10/27/2002 23:55:02.596507 32.245.166.236.65048 > 63.88.212.154.http: P
546101877:546102810(933) ack 3748866716 win 64601 [tos 0x10]
```

Let us determine the likely network topology. This will allow us to set the snort HOME_NET thus enabling snort to be more accurate. The analysis process I have used to determine the network topology is based on the process used by Chris Reining[1]. First we need to determine how many devices snort can see, this is achieved by determining how many unique source and destination MAC addresses are in the capture.

```
Number of unique Source MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.9.27 | awk '{print $2}' |
sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

```
Number of unique Destination MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.9.27 | awk '{print $3}' | sort
-u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

In both cases we only have the same two mac's so it is fair to assume that snort is located between these two network elements. For completeness let us determine which vendor the mac's belong to[2].

```
Extract from IEEE OUI and Company ID Assignments
00-00-0C    (hex)        CISCO SYSTEMS, INC.
00000C      (base 16)    CISCO SYSTEMS, INC.
                         170 WEST TASMAN DRIVE
                         SAN JOSE CA 95134-1706

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

It is unlikely we will be able to determine the actual network addresses belonging to these two mac's however by further analysing flows between them we should be able to determine, with reasonable certainly, our internal network range. To do this we will use a number of Unix utilities to group various fields of each tcpdump.

Below is the anatomy of the first record within the capture. I have highlighted and numbered each import field. This is a handy reference in future grouping commands.

Anatomy of Tcpdump output of first record

```
[root@snort captures]# tcpdump -nner 2002.9.27 -c 1
10:05:29.746507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60:
255.255.255.255.31337 > 32.245.78.8.515: R 0:3(3) ack 0 win 0
```

If we use space as separator we get:

Field 1: Time	**Field 2**: Source MAC
Field 3: Destination MAC	**Field 4**: Ethernet Frame Type
Field 5: Packet Length	**Field 6**: Source IP Port
Field 7: >	**Field 8**: Destination IP Port
Field 9: Flags	**Field 10**: Seq Numbers

First we will determine how many different source addresses originate from each mac address.

Unique Source IP addresses from MAC 0:3:e3:d9:26:c0

```
[root@snort captures]# tcpdump -nner 2002.9.27 "ether src
0:3:e3:d9:26:c0" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
24.167.47.7
<cut>
218.234.199.77
255.255.255.255
[root@snort captures]#

29 in total
```

Unique Source IP addresses from MAC 0:0:c:4:b2:33

```
 [root@snort captures]# tcpdump -nner 2002.9.27 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
32.245.166.119
32.245.166.236
[root@snort captures]#

2 in total
```

Second we will determine how many different destination addresses originate from each mac address.

```
Unique Destination IP addresses from MAC 0:3:e3:d9:26:c0
[root@snort captures]# tcpdump -nner 2002.9.27 "ether src
0:3:e3:d9:26:c0" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
32.245.10.231
32.245.135.248
32.245.141.246
<cut>
32.245.87.215
[root@snort captures]#

134 in total
```

```
Unique Destination IP addresses from MAC 0:0:c:4:b2:33
[root@snort captures]# tcpdump -nner 2002.9.27 "ether src
0:0:c:4:b2:33" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
61.145.114.153
61.145.114.156
<cut>
216.33.240.250
[root@snort captures]#

22 in total
```

From this it would appear that our internal range is 32.245.0.0/16. In addition by looking at the profile of source ip addresses originating from within, it would appear that a natting device is being used (32.245.166.236). This is further confirmed when we observe variation in ttl's originating from this ip address[3].

```
Profile of Source IP addresses
[root@snort captures]# tcpdump -nner 2002.9.27 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq  -c
      3 32.245.166.119
    192 32.245.166.236

Profile of TTL of Source NAT address
[root@snort captures]# tcpdump -nner 2002.9.27 -v "ether src
0:0:c:4:b2:33 and src host 32.245.166.236" | perl -ne 'print "$1\n"
if { $_ =~ /.*\((ttl\s\d.*?)\D/ };' | sort | uniq -c
      8 ttl 122
     64 ttl 124
      8 ttl 125
    112 ttl 240

Where for example 8 is the record count and 122 is the actual ttl.
```

Below is what I believe is the likely network topology extracted from this capture. I have used it as the basis for the rest of the analysis.

```
Basic Network Layout:
** Internet **
   |
******************
*   Cisco Device  *
* 0:3:e3:d9:26:c0 *
******************
   |
****************      *********
* Hub / Switch * <--- * snort *
****************      *********
   |
*****************
* 0:0:c:4:b2:33 *
* Cisco Device  *
*****************
   |
****************** Note This could be done on the above router
*   NAT Device   *
* 32.245.166.236 *
******************
   |
******************
* Rest of Network *
* 32.245.0.0/16   *
******************
```

1.2 Detect was Generated by:

The detect was generated by Snort Version 2.1.2 (Build 25), using a default rule set and a command-line specified HOME_NET.

```
Snort Command Line Used:
[root@snort captures]# snort -b -l /tmp/ -c /etc/snort/snort.conf -r
2002.9.27  -k none -h 32.245.0.0/16
```

The rule, which triggered the alert, is part of the standard snort install and can be found in the file web-misc.rules.

```
The Snort Rule Matched:
[root@snort snort]# grep ".htaccess" *rules
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
MISC .htaccess access"; flow:to_server,established;
content:".htaccess"; nocase; classtype:attempted-recon; sid:1129;
rev:4;)
```

In this rule Snort alert whenever it sees the content of ".htaccess" in a flow to a web server.

1.3 Probability the source address was spoofed:

It is very unlikely that the source address is spoofed. First as this a reconnaissance so the attacker needs to obtain the response. Second for this attempt to work it has to be part of an http connection and thus an established TCP connection. It is very hard to spoof 3-whs to establish a TCP session.

1.4 Description of attack:

Any avid user of the Internet would have on occasions been prompted for a username/password to access a protected document. While there are any number of ways to challenge for a username/password one common way is known as “Basic”. Some web servers, apache[4] for example, commonly store a web username/password in a flat file routinely known as “.htaccess”.

```
Extract of sample ".htaccess" file:
JDOE:rdtgHCcuBiMNU
FRED:v5.MvYkEMeNTY
MARY:dk4MhtlsoUiba
WILMA:NejKl2naRtwkM
```

The format of the file is line-delimited username:encrypted-password. Where, by default, the standard Unix crypt is used to encrypt the password. Since the weak Unix crypt algorithm is used, any standard Unix brute force password cracker can be run against the file. For example an effective tool would be “John the Ripper”[5].

Once the brute force cracker has been run directly against the file the resulting clear text username/passwords can be used to freely access the ‘protected’ documents.

1.5 Attack Mechanism:

When an attacker wishes to obtain unfettered access to a username/password protected site, his number one objective is to obtain any valid username/password. One common way to do this is to obtain a copy of the file that stores all valid usernames/passwords. Due in part to sysadmin’s following documentation to the letter, this file generally has a common name “.htaccess”.

Thus a hacker will commonly try to download the said file so he can brute force it offline. The preference is to download the file, as generally, brute force cracking tools will run much faster offline. In addition the larger log files on the actual web site may cause concern to an alert sysadmin.

The snort alert was triggered as a result of seeing content “.htaccess” in a established flow to a web server.

So in a nutshell we have:
Who: attacker at 210.186.62.136 who is on a reconnaissance mission.
What: wants a copy of .htaccess file which contains username / passwords
Why: so that he can brute force crack them offline
When: 27th October 2002 at 10:45am
Where: a stimulus packet has been sent to our internal web server 32.245.166.119
How: The file can be downloaded and saved using a standard web browser.

1.6 Correlations:

As of 30th May, 2004 dshield.org does not have any reports against this address; remember the capture was taken in 2002.

Brian Coyle a fellow GCIA student has also previously analysed a similar trace[6].

And a reference is the snort signature database.[7]

1.7 Evidence of active targeting:

There are a total of 3 relevant records in the capture. Snort with standard rules only detects two of them. In fact these 3 records are the only occurrence of the source address in all of the capture files contained within the Raw directory.

Tcpdump of all matching source records:

```
[root@snort captures]# tcpdump -nner 2002.9.27  "host 210.186.62.136"
10:45:29.116507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 543:
210.186.62.136.1361 > 32.245.166.119.80: P 805877:806366(489) ack
3123381758 win 8576 (DF)
10:45:29.906507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 590:
210.186.62.136.1361 > 32.245.166.119.80: . 489:1025(536) ack 793 win
8576 (DF)
10:55:28.146507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 1075:
210.186.62.136.1437 > 32.245.166.119.80: P
3774972488:3774973509(1021) ack 1430629 win 32696 [tos 0x10]
```

The second record is a request of the original page requested in the first record with an invalid basic authentication record attached; it resulted in the following snort alert. It was transmitted some 790mSec latter. This would probably have been in response to receiving a 401 error code [8].

Snort Alert generated by second record:

```
[**] [1:1260:6] WEB-MISC long basic authorization string [**]
[Classification: Attempted Denial of Service] [Priority: 2]
10/27-10:45:29.906507 210.186.62.136:1361 -> 32.245.166.119:80
TCP TTL:108 TOS:0x0 ID:33581 IpLen:20 DgmLen:576 DF
***A**** Seq: 0xC4DDE  Ack: 0xBA2B0916  Win: 0x2180  TcpLen: 20
[Xref => http://www.securityfocus.com/bid/3230]
```

The third record, although snort isn't concerned, is interesting for a couple of reasons. First it has the unusual ip.id of zero. Second is that it's ttl is 240 where as the two previous records were 108. The main difference is that it is trying to download "logos.html" instead of ".htaccess" in the same directory structure.

I believe it is active targeting. The fact that ".htaccess" was requested is enough in itself, coupled with different ttl then something strange is going on.

1.8 Severity:

Criticality: 4 It is hard to determine how important this asset is without knowing exactly what the business purpose of the server is, that aside it is an internet connected web server with some form of password-protected documents. Thus by inference it must be reasonably important.

Lethality: 5 If this file is obtained the impact can be disastrous. Other alerts from the capture file indicate that this server is a Redhat Linux server running Apache with Frontpage extensions. In addition the second and third records have basic authentication credentials embedded in the requests inferring that authentication is needed to access this server.

System Countermeasures: 2, This is very hard to determine without knowledge of how well the server has been maintained. Since I can find no alerts originating from this server except 68 “Attack- Responses 403 Forbidden” one can assume it has been installed and maintained correctly. That was until one takes into consideration the version of Apache. The “403 messages” indicated Apache 1.3.12 is being used. A quick google indicates Apache was officially up to version 1.3.27 as at Oct 3 2002 [9]. I have assumed that the syadmin has not over-ridden the http response header.

Network Countermeasures: 1, http is a permitted protocol to this host and thus the web server has been afforded no protection by firewall or border routers.

(4+5) – (2+1) = 6

1.9 Defensive Recommendations:

As outlined in the Snort signature database, one should first determine if the page can be successfully downloaded. If it can, then escalate the event to an incident and manage.

One should then ensure all servers within your organisation are configured correctly to restrict access to any files containing username/passwords. The snort signature database indicates how to do it for files starting with “.ht” Use it as a basis for blocking any relevant files.

```
Preventing Access to .ht* files in Apache:
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

It generally is also be more effective to offload authentication and authorisation via suitable apache modules e.g. via radius or ldap.

If you continue to use apache’s htpasswd to manage user password’s, wrap the htpasswd in a suitable script / alias to ensure that it is always called with the ‘-m’ command-line option thus enforcing the use of the more secure MD5 encryption algorithm.

Also, update to the latest version of apache.

1.10 Multiple Choice Question:

Why should an intrusion analyst working for an eCommerce site using Apache be concerned about the following alert. ?

[**] [1:1129:4] WEB-MISC .htaccess access [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/27-10:45:29.116507 210.186.62.136:1361 -> 32.245.166.119:80
TCP TTL:108 TOS:0x0 ID:29485 IpLen:20 DgmLen:529 DF
AP Seq: 0xC4BF5 Ack: 0xBA2B05FE Win: 0x2180 TcpLen: 20

(a) doesn’t need to worry as the encryption is very strong.

(b) as the files contains all web usernames and weakly encrypted passwords.
(c) apache never uses this file so this is not a concern.
(d) ".htaccess" files only contain usernames and as such is not that valuable.

The best answer is (b).
(a) is wrong, as historically encryption was based on crypt.
(c) is wrong, as ".htaccess" files have commonly been used to control access to files and directories on apache web server.
(d) is wrong as it is a line-delimited file containing username and encrypted passwords.

1.11 Excerpts from Intrusion's Discussion group:

Originally Submitted:
mtvpm mtvpm at bigpond.com
Tue Jun 1 14:46:38 UTC 2004

My (only) Response to Questions:
Subject: Re: [Intrusions] LOGS: GIAC GCIA Version 3.4 Practical Detect Michael Meacle
Date: Sat, 19 Jun 2004 02:16:34 +1000
From: mtvpm mtvpm@bigpond.com
Reply-To: "Intrusions List (GCIA Practicals)" intrusions@lists.sans.org
To: intrusions@lists.sans.org

Mohan,

Thanks for the questions.

1.Sorry for being a little too terse thus your need to highlight a gap in my reasoning, not to mention a error in my document which I found while trying to check my logic.

First my error, In my numbering of the fields I didn't count the '>', this would confuse you if you were looking back at its field references. See below the amended field numbering.

```
        Field 8: Destination IP Port
        Field 9: Flags
        Field 10: Seq Numbers
```

Now as for how I concluded that only two host were 'effectively' talking to the internet is based on analysing the flows. Least just consider the 32.245.x range.

- We have 2 (Source IP Addresses) with a destination mac of 0:3:e3:d9:26:c0. (remember we only had two mac's so we can infer that the destination mac address is the above because it can't be 0:0:c:4:b2:33 - a review at the raw data confirms this)
- We also have 134 (Destination IP Addresses) with a source mac of 0:3:e3:d9:26:c0.

So by focusing only on the one mac address we can see only 2 addresses going to, yet 134 coming from it. The primary reason why we see 134 incoming address I believe is we own all 65535 (based on the fact that 134 addresses were randomly spaced across the B'class range) address and over the capture period 134 were scanned from the internet. In addition the reason we only see 2 addresses flowing in both directions (in this capture) is they (at present) are the only valid (active) i/c and o/g real

addresses.

2. While trying to analyse the raw records around the records of interest I noticed the variation. The primary reason I noticed the variation was that I was trying to verify the OS (against the 304 error messages [NOTE MTM: 23/06/2004 that should have said 403]) so that I could compare it against well known values as outlined in the Sans training material "Network Traffic Analysis - Using Tcpdump parts 1 & 2 p4-24&25. If you don't have access to the material have a look at http://project.honeynet.org/papers/finger/traces.txt .

When I noticed the variation I had to explain it. The only two ideas I could come up with were NAT or crafted packets. As I believed it unlikely the file would have been full of crafted packets, the fact that I could see so much traffic from the one host and the fact that NAT'ng is commonly done by enterprises to hide their real internal addresses. I played the odds.

I wasn't sure what should happen at a nat'ng device, or in fact whether they all act the same (unlikely), e.g.
- do they set the ttl to some predefined value e.g. to values as outlined in the honeynet paper
- do they decrement by 1
- do they leave it alone
I didn't know but I went with the second one as I believed it to be the safest against asymmetrical routing loops. As such it was the only way I could explain the 4 grouped ttl values.

As a result of your question I consulted google ..... again. I have found a very interesting article by Peter Phaal (http://www.sflow.org/detectNAT/) which outlines his concept of detecting unauthorised NAT devices. His article (thankfully) confirms that what I observed is probably nat'ng. In his article he refers to another article by Steven Bellovin (http://www.research.att.com/~smb/papers/fnat.pdf) it outlines a further (some what more complex) way of doing it based on ip-id. Due to the fact I'm running very late on my assignment I have only skimmed both articles but both appear to confirm that nat'ng was taking place.

I'm glad you challenged so now we both know two new ways.

Out of interest while doing my other detect's I refined the way of counting each TTL's as on my 3rd detect (due to some strange packets) I was getting erratic results. For you convenience here is the better command.

```
        tcpdump -nner 2002.9.27 -v "ether src 0:0:c:4:b2:33
and src host 32.245.166.236" | perl -ne 'print"$1\n" if { $_ =~
/.*\((ttl\s\d.*?)\D/ };' | sort | uniq -c
        8 ttl 122
        64 ttl 124
        8 ttl 125
        112 ttl 240
```

Hope that helps
Mick

Mohan Chirumamilla wrote:

Michael,

```
May be it is obvious, but I am missing something here and would appreciate
if you could help me out with the following two questions.

1.  Your analysis shows that there are possibly two hosts 32.245.166.119
and 32.245.166.236 talking to the Internet.

Unique Source IP addresses from MAC 0:0:c:4:b2:33
[root at snort captures]# tcpdump -nner 2002.9.27 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3 "."
$4}' | sort -n | uniq
  32.245.166.119
  32.245.166.236
[root at snort captures]#

And here, the numbers show that there are more than two hosts
participating from within the 32.245.0.0 address range.

Unique Destination IP addresses from MAC 0:3:e3:d9:26:c0
[root at snort captures]# tcpdump -nner 2002.9.27 "ether src
0:3:e3:d9:26:c0" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
  32.245.10.231
  32.245.135.248
  32.245.141.246
  <cut>
  32.245.87.215
[root at snort captures]#

In a normal TCP session, traffic flows in either direction almost
symmetrically.  What I meant by symmetric is that, for every data packet
received the receipent sends an ack back to the sender (assuming that
there's no loss of packets at all).
So that being said, should,'t we be seeing almost the same hosts on each
of your lists you described above.  I agree that it might not be the case
if we are talking about UDP.  I did not take a look at the dump yet.  But
to me that case is a bit unusual.

One possible reason could be is that all hosts in the second list could be
sitting in between the two routers.  But having 134 hosts in a "DMZ" like
set-up is again....little bit unusual.

2.  You derived through your analysis that there could be a NAT'ing device
(32.245.166.236).  Can you please throw some more details on how the study
of TTL field contributed to your decission?  Based on your analysis, and
assuming that my "possible reason" that I mentioned above is ture.....I
got an impression that the source IP addresses are not being modified
during the NAT'ing process.  Is this correct?
```

Reference:

[1] Reining, Chris
LOGS: GIAC GCIA Version 3.4 Practical Detect Chris Reining, (8/2/2004)
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00064.html (28/5/2004)

[2] IEEE OUI and Company_id Assignments
URL: http://standards.ieee.org/regauth/oui/oui.txt (30/5/2004)

[3] Phaal, Peter.
Detection NAT Devices using sFlow

URL: http://www.sflow.org/detectNAT/ (19/06/2004)

[4] Apache HTTP Server 2.0,
Authentication, Authorization and Access Control.
URL: http://httpd.apache.org/docs-2.0/howto/auth.html (30/5/2004)

[5] John the Ripper password cracker
URL: http://www.openwal.com/john (30/5/2004)

[6] Coyle, Brian.
GCIA Practical V3.1 Part 1 – State of IDS (April 2002)
URL: http://www.giac.org/practical/GCIA/Brian_Coyle_GCIA.pdf (30/5/2004)

[7] Sourcefire Research Team,
Snort Signature Database.
URL: http://www.snort.org/snort-db/sid.html?sid=1129 (30/5/2004)

[8] Part of Hypertext Transfer Protocol – HTTP/1.1
URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html (23/06/2004)

[9] Jagielski, Jim.
Apache 1.3.27 Released (3/10/2002)
URL: http://archives.neohapsis.com/archives/apache/2002/0019.html (30/5/2004)

2. Detect #2 nimda

```
Extract from /tmp/alert
[**] [1:1002:5] WEB-IIS cmd.exe access [**]
[Classification: Web Application Attack] [Priority: 1]
11/04-09:30:42.666507 140.116.141.73:4156 -> 207.166.8.195:80
TCP TTL:109 TOS:0x0 ID:39971 IpLen:20 DgmLen:136 DF
***AP*** Seq: 0x8E75B600  Ack: 0x3A381913  Win: 0xFD20  TcpLen: 20
```

2.1 Source Of Trace:

The raw log file was obtained from http://www.incidents.org/logs/Raw/2002.10.3 . As numerous students have previously pointed out before the logfile timestamp and records contained within don't match. In fact according to tcpdump the actual period is for 03/11/2002.

```
Determining range of records in the capture
[root@snort captures]# tcpdump -nn -r 2002.10.3 -tttt
11/03/2002 00:00:49.426507 216.77.219.195.48839 > 207.166.233.11.1080:
S 1769720 505:1769720505(0) win 1024
<cut>
11/03/2002 23:54:58.496507 209.226.144.25.3914 > 207.166.87.53.139: P
4018084:4018143(59) ack 2276636370 win 8572 NBT Packet (DF)
```

Lets determine the likely network topology. This will allow us to set the snort HOME_NET thus enabling snort to be more accurate. The analysis process I have used to determine the network topology is based on the process used by Chris Reining[1]. First we need to determine how many devices snort can see, this is achieved by determining how many unique source and destination MAC addresses are in the capture.

```
Number of unique Source MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.10.3 | awk '{print $2}' |
sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

```
Number of unique Destination MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.10.3 | awk '{print $3}' | sort
-u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

In both cases we only have the same two mac's so it is fair to assume that snort is located between these two network elements. For completeness let us determine which vendor the mac's belong too[2].

```
Extract from IEEE OUI and Company ID Assignments
00-00-0C    (hex)        CISCO SYSTEMS, INC.
00000C      (base 16)    CISCO SYSTEMS, INC.
                         170 WEST TASMAN DRIVE
                         SAN JOSE CA 95134-1706
00-03-E3    (hex)        Cisco Systems, Inc.
0003E3      (base 16)    Cisco Systems, Inc.
                         170 West Tasman Dr.
                         San Jose CA 95134
                         UNITED STATES
```

It is unlikely we will be able to determine the actual network addresses belonging to these two mac's however by further analysing flows between them we should be able to determine, with reasonable certainly, our internal network range. To do this we will use a number of Unix utilities to group various fields of each tcpdump.

Below is the anatomy of the first record within the capture. I have highlighted and numbered each import field. This is a handy reference in future grouping commands.

Anatomy of Tcpdump output of first record

```
[root@snort captures]# tcpdump -nner 2002.10.3 -c 1
10:00:49.426507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60:
216.77.219.195.48839 > 207.166.233.11.1080: S
1769720505:1769720505(0) win 1024

If we use space as separator we get:
```

Field 1: Time	**Field 2**: Source MAC
Field 3: Destination MAC	**Field 4**: Ethernet Frame Type
Field 5: Packet Length	**Field 6**: Source IP Port
Field 7: >	**Field 8**: Destination IP Port
Field 9: Flags	**Field 10**: Seq Numbers

First we will determine how many different source addresses originate from each mac address.

Unique Source IP addresses from MAC 0:3:e3:d9:26:c0

```
[root@snort captures]# tcpdump -nner 2002.10.3 "ether src
0:3:e3:d9:26:c0" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
24.165.67.204
61.223.29.184
<cut>
218.14.156.61
255.255.255.255
[root@snort captures]#

38 in total
```

Unique Source IP addresses from MAC 0:0:c:4:b2:33

```
 [root@snort captures]# tcpdump -nner 2002.10.3 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
207.166.87.157
207.166.87.40
[root@snort captures]#

2 in total
```

Second we will determine how many different destination addresses originate from each mac address.

```
Unique Destination IP addresses from MAC 0:3:e3:d9:26:c0
[root@snort captures]# tcpdump -nner 2002.10.3 "ether src
0:3:e3:d9:26:c0" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
207.166.0.20
207.166.0.204
<cut>
207.166.99.101
207.166.99.151
[root@snort captures]#

911 in total
```

```
Unique Destination IP addresses from MAC 0:0:c:4:b2:33
[root@snort captures]# tcpdump -nner 2002.10.3 "ether src
0:0:c:4:b2:33" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
4.33.9.31
4.60.214.29
<cut>
217.224.151.162
218.186.94.120
[root@snort captures]#

800 in total
```

From this it would appear that our internal range is 207.166.0.0/16. In addition by looking at the profile of source ip addresses originating from within it would appear that a nating device is being used (207.166.87.157). This is further confirmed when we observe variation in ttl's originating from this ip address[3].

```
Profile of Source IP addresses
[root@snort captures]# tcpdump -nner 2002.10.3 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq  -c
   1048 207.166.87.157
      3 207.166.87.40

Profile of TTL of Source NAT address
[root@snort captures]#  tcpdump -nner 2002.10.3 -v "ether src
0:0:c:4:b2:33 and src host 207.166.87.157" | perl -ne 'print "$1\n"
if { $_ =~ /.*\((ttl\s\d.*?)\D/ };' | sort | uniq -c
    975 ttl 124
     73 ttl 240

Where for example 975 is the record count and 124 is the actual ttl.
```

Below is what I believe is the likely network topology extracted from this capture. I have used it as the basis for the rest of the analysis.

```
Basic Network Layout:
** Internet **
   |
******************
*   Cisco Device  *
* 0:3:e3:d9:26:c0 *
******************
   |
****************      *********
* Hub / Switch * <--- * snort *
****************      *********
   |
*****************
* 0:0:c:4:b2:33 *
* Cisco Device  *
*****************
   |
****************** Note This could be done on the above router
*   NAT Device   *
* 207.166.87.157 *
******************
   |
********************
* Rest of Network  *
* 207.166.0.0/16   *
********************
```

2.2 Detect was Generated by:

The detect was generated by Snort Version 2.1.2 (Build 25), using a default rule set and a command-line specified HOME_NET.

```
Snort Command Line Used:
[root@snort captures]# snort -b -l /tmp/ -c /etc/snort/snort.conf -r
2002.10.3  -k none -h 207.166.0.0/16
```

The rule, which triggered the alert, is part of the standard snort install and can be found in the file web-iis.rules.

```
The Snort Rule Matched:
[root@snort snort]# grep "WEB-IIS cmd.exe access" *rules
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
IIS cmd.exe access"; flow:to_server,established; content:"cmd.exe";
nocase; classtype:web-application-attack; sid:1002;  rev:5;)
```

In this rule Snort alerts whenever it sees the content of “cmd.exe” in a flow to a web server. A full dump of the packet triggering the alert is shown below.

Tcpdump of two of eight packets triggering the alert:

```
[root@snort captures]# tcpdump -nner 2002.10.3 -X -v " host
140.116.141.73 and host 207.166.8.195 and port 80"
09:30:42.666507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 150:
140.116.141.73.4156 > 207.166.8.195.80: P [bad tcp cksum 79c5!]
2390078976:2390079072(96) ack 976754963 win 64800 (DF) (ttl 109, id
39971, len 136, bad cksum cb6e!)
0x0000 4500 0088 9c23 4000 6d06 cb6e 8c74 8d49   E....#@.m..n.t.I
0x0010 cfa6 08c3 103c 0050 8e75 b600 3a38 1913   .....<.P.u..:8..
0x0020 5018 fd20 1b9e 0000 4745 5420 2f73 6372   P.......GET./scr
0x0030 6970 7473 2f2e 2e25 3563 2e2e 2f77 696e   ipts/..%5c../win
0x0040 6e74 2f73 7973 7465 6d33 322f 636d 642e   nt/system32/cmd.
0x0050 6578 653f 2f63 2b64 6972 2072 2048 5454   exe?/c+dir.r.HTT
0x0060 502f 312e 300d 0a48 6f73 743a 2077 7777   P/1.0..Host:.www
0x0070 0d0a 436f 6e6e 6e65 6374 696f 6e3a 2063   ..Connnection:.c
0x0080 6c6f 7365 0d0a 0d0a                       lose....
<cut>
09:32:34.886507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 171:
140.116.141.73.3263 > 207.166.8.195.80: P [bad tcp cksum 9819!]
0:117(117) ack 1 win 64800 (DF) (ttl 109, id 46786, len 157, bad
cksum b0ba!)
0x0000 4500 009d b6c2 4000 6d06 b0ba 8c74 8d49   E.....@.m....t.I
0x0010 cfa6 08c3 0cbf 0050 9298 9d2d 14f9 b09d   .......P...-....
0x0020 5018 fd20 07c8 0000 4745 5420 2f5f 7674   P.......GET./_vt
0x0030 695f 6269 6e2f 2e2e 2535 632e 2e2f 2e2e   i_bin/..%5c../..
0x0040 2535 632e 2e2f 2e2e 2535 632e 2e2f 7769   %5c../..%5c../wi
0x0050 6e6e 742f 7379 7374 656d 3332 2f63 6d64   nnt/system32/cmd
0x0060 2e65 7865 3f2f 632b 6469 7220 632b 6469   .exe?/c+dir.c+di
0x0070 7220 4854 5450 2f31 2e30 0d0a 486f 7374   r.HTTP/1.0..Host
0x0080 3a20 7777 770d 0a43 6f6e 6e6e 6563 7469   :.www..Connnecti
0x0090 6f6e 3a20 636c 6f73 650d 0a0d 0a          on:.close....

The yellow highlight being the most important.
```

2.3 Probability the source address was spoofed:

It is very unlikely that the source address is spoofed. It is either a nimda infected host or an attacker with a script trying to find a vulnerable host e.g. their next victim. In either case they need to be able to see the response. Second for this attempt to work it has to be part of an http connection and thus an established TCP connection. It is very hard to spoof 3-whs to establish a TCP session.

2.4 Description of attack:

In the capture we see two different attempts by an infected host to send a stimulus probe to our internal web server. The http requests are distinct in that they try to take advantage of a known vulnerability in unpatched IIS servers, version 4 and 5. The vulnerability known as “Unicode Web Traversal exploit” [4] relies on the interaction of canonicalization and Unicode to allow url’s, which would normally be prevented by IIS because they contained “../..”, to be serviced by the server. To do this we need to replace (as a minimum) any ‘/’ characters with its equivalent Unicode, for example replace the ‘/’ with ‘%5c’. This allows a savvy user (worm) to run arbitrary commands on the web server as user IUSR_machinename.

In this particular case we have another infected host running the reconnaissance probes to determine if our web server is vulnerable. If from the responses it determines the host is vulnerable it will get the “new victim” to download, via tftp, a

copy of the worm code. Using the same mechanism it will install and activate the worm.

2.5 Attack Mechanism:

A server infected with the worm will attempt to infect as many other computers as possible. It will do this using a number of mechanisms including mass mailing, network share propagation, and Unicode web transversal vulnerability.

So in a nutshell we have:
Who: attacker/infected host at 140.116.141.73
What: sends a couple of stimulus packets to our web server 207.166.8.195
Why: so that they can determine if the web server is vulnerable to nimda worm.
When: 04th November 2002 at 9:30am
Where: to our internal web server 207.166.8.195
How: The worm will do it automatically. If it is a hacker (unlikely) he must be using a script as both records are only seconds apart.

2.6 Correlations:

As of 4th June, 2004 dshield.org does not have any reports against this address; remember the capture was taken in 2002.

Danny Li, a fellow GCIA student, has also previously analysed a similar nimda detect[5].

Cert advisory also explains in detail how nimda operates, including the very important expected log entries, which assisted me in determining that it is indeed part of a nimda trace [6].

And a reference is the snort signature database.[7]

2.7 Evidence of active targeting:

The nimda worm randomly scans for the next vulnerable host so in this case we are merely a statistic. However it is interesting that on 8th November we receive more random nimda scans from 140.116.141.108 to other internal hosts.

2.8 Severity:

Criticality: 5, Very hard to determine how important this asset is without knowing exactly what the business purpose of the server is, that aside, it is an internet connected web server. Thus, by inference, it must be reasonably important.

Lethality: 5, An administrative shell is obtainable if we are vulnerable.

System Countermeasures: 5, Very hard to determine without knowledge of how well the server has been maintained. Since I can find no subsequent alerts from our internal web server I believe either it is not running IIS or is not vulnerable.

Network Countermeasures: 1, http is a permitted protocol to this host and thus the web server has been afforded no protection by firewall or border routers.

(5+5) – (5+1) = 4

2.9 Defensive Recommendations:

One should ensure all servers within your organisation are configured correctly and fully patched.

Should also ensure that the default install and sample scripts are removed.

In addition it is important to install the web server on a different partition than the OS.

2.10 Multiple Choice Question:

If someone requested the following url from an unpatched IIS 4.0 server, what happens.

http://vulnerable.com/scripts/..%5c../winnt/system32/cmd.exe /c dir

(a) the default web page will be displayed.
(b) will spawn a cmd shell, run the command "dir" in the shell, return the output of the command to the user's browser and then terminate the shell.
(c) redirect the user to http://www.cmd.com/
(d) the server returns error as it doesn't understand %5c.

The best answer is (b).
(a) is wrong, see (b).
(c) is wrong, see (b).
(d) is wrong, IIS servers understand the Unicode '%5c'.

Reference:

---

[1] Reining, Chris
LOGS: GIAC GCIA Version 3.4 Practical Detect Chris Reining, (8/2/2004)
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00064.html (28/5/2004)

[2] IEEE OUI and Company_id Assignments
URL: http://standards.ieee.org/regauth/oui/oui.txt (30/5/2004)

[3] Phaal, Peter.
Detection NAT Devices using sFlow
URL: http://www.sflow.org/detectNAT/ (19/06/2004)

[4] Microsoft Security Bulletin (MS00-078) (Oct 17th 2000)
URL: http://www.microsoft.com/technet/security/bulletin/MS00-078.mspx (4/6/2004)

[5] Li, Danny.
LOGS: GIAC GCIA Version 3.3 Practical Detect 1/3 (22/10/2003)
URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00149.html (1/6/2004)

[6] CERT Advisory CA-2001-26 Nimda Worm. (September 25,2001)
URL: http://www.cert.org/advisories/CA-2001-26.html (4/6/2004)

[7] Sourcefire Research Team,
Snort Signature Database.
URL: http://www.snort.org/snort-db/sid.html?sid=1002 (30/5/2004)

3. Detect #3 FTP command overflow attempt

```
Extract from /tmp/alert
[**] [1:1748:4] FTP command overflow attempt [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]
05/17-02:01:19.264488 212.164.216.3:10071 -> 78.37.212.165:21
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:173
***AP*** Seq: 0x2413567D  Ack: 0xBE5BCB77  Win: 0x7D78  TcpLen: 20
[Xref => http://www.securityfocus.com/bid/4638]
```

3.1 Source Of Trace:

The raw log file was obtained from http://www.incidents.org/logs/Raw/2002.4.16 . As numerous students have previously pointed out before the logfile timestamp and records contained within don't match. In fact according to tcpdump the actual period is for 16/5/2002.

```
Determining range of records in the capture
[root@snort captures]# tcpdump -n -r 2002.4.16 -tttt | more
05/16/2002 00:05:52.074488 207.229.152.8.http > 78.37.212.28.62487: P
4269347772:4269349232(1460) ack 3957273 win 32120 (DF)
<cut>
05/16/2002 23:53:40.944488 207.178.214.185.knetd > 78.37.212.165.ftp: P
2256582263:2256582279(16) ack 491506774 win 8217 (DF)
```

Let us determine the likely network topology. This will allow us to set the snort HOME_NET thus enabling snort to be more accurate. The analysis process I have used to determine the network topology is based on the process used by Chris Reining[1]. First we need to determine how many devices snort can see, this is achieved by determining how many unique source and destination MAC addresses are in the capture.

```
Number of unique Source MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.4.16 | awk '{print $2}' |
sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

```
Number of unique Destination MAC addresses in capture
[root@snort captures]# tcpdump -ner 2002.4.16 | awk '{print $3}' | sort
-u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

In both cases we only have the same two mac's so it is fair to assume that snort is located between these two network elements. For completeness let us determine which vendor the mac's belong too[2].

```
Extract from IEEE OUI and Company ID Assignments
00-00-0C    (hex)        CISCO SYSTEMS, INC.
00000C      (base 16)    CISCO SYSTEMS, INC.
                         170 WEST TASMAN DRIVE
                         SAN JOSE CA 95134-1706
00-03-E3    (hex)        Cisco Systems, Inc.
0003E3      (base 16)    Cisco Systems, Inc.
                         170 West Tasman Dr.
                         San Jose CA 95134
                         UNITED STATES
```

It is unlikely we will be able to determine the actual network addresses belonging to these two mac's however by further analysing flows between them we should be able to determine, with reasonable certainly, our internal network range. To do this we will use a number of Unix utilities to group various fields of each tcpdump.

Below is the anatomy of the first record within the capture. I have highlighted and numbered each import field. This is a handy reference in future grouping commands.

Anatomy of Tcpdump output of first record

```
[root@snort captures]# tcpdump -nner 2002.4.16 -c 1
10:05:52.074488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 1514:
207.229.152.8.80 > 78.37.212.28.62487: P 4269347772:4269349232(1460)
ack 3957273 win 32120 (DF)
```

If we use space as separator we get:

Field 1: Time	**Field 2**: Source MAC
Field 3: Destination MAC	**Field 4**: Ethernet Frame Type
Field 5: Packet Length	**Field 6**: Source IP Port
Field 7: >	**Field 8**: Destination IP Port
Field 9: Flags	**Field 10**: Seq Numbers

First we will determine how many different source addresses originate from each mac address.

Unique Source IP addresses from MAC 0:3:e3:d9:26:c0

```
[root@snort captures]# tcpdump -nner 2002.4.16 "ether src
0:3:e3:d9:26:c0" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
4.3.95.238
12.37.117.126
<cut>
218.96.62.2
255.255.255.255
[root@snort captures]#

108 in total
```

Unique Source IP addresses from MAC 0:0:c:4:b2:33

```
 [root@snort captures]# tcpdump -nner 2002.4.16 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
78.37.212.165
78.37.212.28
[root@snort captures]#

2 in total
```

Second we will determine how many different destination addresses originate from each mac address.

```
Unique Destination IP addresses from MAC 0:3:e3:d9:26:c0
[root@snort captures]# tcpdump -nner 2002.4.16 "ether src
0:3:e3:d9:26:c0" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "."
$3 "." $4}' | sort -n | uniq
78.37.0.113
78.37.0.76
<cut>
78.37.96.218
78.37.98.99
[root@snort captures]#

120 in total
```

```
Unique Destination IP addresses from MAC 0:0:c:4:b2:33
[root@snort captures]# tcpdump -nner 2002.4.16 "ether src
0:0:c:4:b2:33" | awk '{print $8}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq
4.42.79.213
12.109.100.230
<cut>
216.33.240.250
217.211.124.172
[root@snort captures]#

114 in total
```

From this it would appear that our internal range is 78.37.0.0/16. In addition by looking at the profile of source ip addresses originating from within it would appear that a nating device is being used (78.37.212.28). This is further confirmed when we observe variation in ttl's originating from this ip address[3].

```
Profile of Source IP addresses
[root@snort captures]# tcpdump -nner 2002.4.16 "ether src
0:0:c:4:b2:33" | awk '{print $6}' | awk -F\. '{print $1 "." $2 "." $3
"." $4}' | sort -n | uniq  -c
      9 78.37.212.165
   2961 78.37.212.28

Profile of TTL of Source NAT address
[root@snort captures]# tcpdump -nner 2002.4.16 -v "ether src
0:0:c:4:b2:33 and src host 78.37.212.28" | perl -ne 'print "$1\n" if
{ $_ =~ /.*\((ttl\s\d.*?)\D/ };' | sort | uniq -c
     13 ttl 123
    890 ttl 124
    476 ttl 125
   1582 ttl 240

Where for example 13 is the record count and 123 is the actual ttl.
```

Below is what I believe is the likely network topology extracted from this capture. I have used it as the basis for the rest of the analysis.

```
Basic Network Layout:
** Internet **
   |
******************
*   Cisco Device  *
* 0:3:e3:d9:26:c0 *
******************
   |
****************      *********
* Hub / Switch * <--- * snort *
****************      *********
   |
******************
* 0:0:c:4:b2:33 *
* Cisco Device  *
******************
   |
****************** Note This could be done on the above router
*   NAT Device  *
* 78.37.212.28  *
******************
   |
********************
* Rest of Network *
* 78.37.0.0/16    *
********************
```

3.2 Detect was Generated by:

The detect was generated by Snort Version 2.1.2 (Build 25), using a default rule set and a command-line specified HOME_NET.

```
Snort Command Line Used:
[root@snort captures]# snort -b -l /tmp/ -c /etc/snort/snort.conf -r
2002.4.16  -k none -h 78.37.0.0/16
```

The rule, which triggered the alert, is part of the standard snort install and can be found in the file “ftp.rules".

```
The Snort Rule Matched:
[root@snort snort]# grep "FTP command overflow attempt" *rules
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP command
overflow attempt"; flow:to_server,established,no_stream; dsize:>100;
reference:bugtraq,4638; classtype:protocol-command-decode; sid:1748;
rev:4;)
```

In this rule Snort alerts whenever the payload size is greater than 100 in a flow to a FTP command port. A full dump of the packet triggering the alert is shown below and shows that the payload was 133 bytes.

Tcpdump payload of the packet triggering the alert:

```
[root@snort captures]# tcpdump -nner 2002.4.16 -X -v " host
212.164.216.3 and 78.37.212.165 and port 21 and ip[4:2] == 0"
02:01:19.264488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 187:
212.164.216.3.10071 > 78.37.212.165.21: P [bad tcp cksum 8956!]
605247101:605247234(133) ack 3193686903 win 32120 [tos 0x10]  (ttl
240, id 0, len 173, bad cksum 0!)
0x0000 4510 00ad 0000 0000 f006 0000 d4a4 d803   E...............
0x0010 4e25 d4a5 2757 0015 2413 567d be5b cb77   N%..'W..$.V}.[.w
0x0020 5018 7d78 0000 0000 5553 4552 2061 6e6f   P.}x....USER.ano
0x0030 6e79 6d6f 7573 0d0a 5041 5353 2079 6f75   nymous..PASS.you
0x0040 726e 616d 6540 796f 7572 636f 6d70 616e   rname@yourcompan
0x0050 792e 636f 6d0d 0a52 4553 5420 3130 300d   y.com..REST.100.
0x0060 0a52 4553 5420 300d 0a54 5950 4520 410d   .REST.0..TYPE.A.
0x0070 0a50 4153 560d 0a4c 4953 5420 2f70 7562   .PASV..LIST./pub
0x0080 2f75 7362 2f67 616d 6570 6f72 742e 7064   /usb/gameport.pd
0x0090 660d 0a54 5950 4520 490d 0a50 4153 560d   f..TYPE.I..PASV.
0x00a0 0a52 4554 5220 2f70 7562 2f75 73          .RETR./pub/us

The yellow highlight being the most important (shortly).
And ip id of 0 is unusual.
```

3.3 Probability the source address was spoofed:

It is very unlikely that the source address is spoofed. First this is part of an established ftp-command channel. Second for this attempt to work it has to be part of an ftp connection and thus a established TCP connection. It is very hard to spoof 3-whs to establish a TCP session.

3.4 Description of attack:

The alert rule was written as a result of a vulnerability in a free Windows 32 bit based ftp server[4]. The 3Com demon is vulnerable to a buffer overflow when the server receives 400 or more characters. The snort developers have conservatively set a low value of 100 to detect invalid ftp commands. Clearly the above capture will not cause a buffer overflow and consequently a DOS.

Exploit DOS code has also been posted to bugtraq, however a quick review of the code indicates that it would send 420 “A” and as such has not been used here.

So what is the attack?
As far as I can establish is it a long and invalid ftp command. It is invalid for a couple of reasons. The first (highlighted in yellow above) is the fact that the first REST command is not immediately followed with a FTP service command as required in the rfc959[5]. The second reason is that the rfc959 states that communication between user and server is intended to be an alternating dialogue and as such a user protocol interpreter should wait for a response to any command before sending a new command. There are at least 9 consecutive commands in the above record, hardly an alternating dialogue.

I tried to identify if any client would naturally generate such an invalid request. Since I couldn’t identify one I suggest it is a crafted, yet invalid, ftp request. A review of the payload indicated it had no malicious payload.

3.5 Attack Mechanism:

In a nutshell we have:
Who: attacker at 212.164.216.3
What: attempting an invalid stimulus against an internal FTP server 78.37.212.165
Why: He could have number motives

- Denial of Service, unlikely as it isn't big enough for the known 3Com vulnerability.
- Attempting to get malicious code to execute, this is unlikely as the payload doesn't appear to have any malicious content.
- Reconnaissance to try and identify the ftp server's response to a incorrectly constructed request (stimulus), this is the most likely.

When: 17th May 2002 at 2:01:19 am
Where: from a internal ftp server 78.37.212.165
How: unknown client in fact it would appear to be a crafted packet.

3.6 Correlations:

As of 30th May, 2004 dshield.org does not have any reports against the address 212.164.216.3; remember the capture was taken in 2002.

And a reference is the snort signature database.[6]

Bugtraq also details vulnerability in 3Com's windows ftp daemon (3Cdaemon)[4].

3.7 Evidence of active targeting:

There are a total of 12 records between these two hosts. Ten were immediately before this record and one immediately after. The other 11 records appeared to be valid anonymous ftp attempts.

Considering that the payload violates rfc959, and I can find no standard client to generate such a request I believe it to be a calculated stimulus attack.

So yes, there is evidence of active targeting.

3.8 Severity:

Criticality: 3 It is hard to determine how important this asset is without knowing exactly what its business purpose of the server is, that aside it is an Internet connected ftp and www server. It would also appear that anonymous ftp access is allowed. Thus by inference one would believe it to be some form of vendor supplied informational site.

Lethality: 1 This is a stimulus to gather information about the ftp server.

System Countermeasures: 3, Very hard to determine without knowledge of how well the server has been maintained. There are however a lot of "Attack- Responses 403 Forbidden". The "403 messages" indicated Apache 1.3.12 is being used. A quick google indicates Apache was officially up to version 1.3.27 as at Oct 3 2002 [7]. I have assumed that the syadmin has not over-ridden the http response header. If the http demon is so out-of-date one has to assume the ftp demon is also un-maintained.

Network Countermeasures: 1, ftp is a permitted protocol to this host and thus the web server has been afforded no protection by firewall or border routers.

(3+1) – (3+1) = 0

3.9 Defensive Recommendations:

The packet appears to be a specially crafted reconnaissance probably to confirm the actual version of ftp server. As such it is important that a suitably qualified sysadmin ensure the server is fully patched and correctly configured.

Since the real motive of the attacker cannot be conclusively determined with the above capture, I suggest the analyst temporarily add the following rule to the local.rules file. Remember, it will now have the highest precedence and in particular will have a higher priority than all other rules in the ftp.rules file. If this creates an issue, you probably should modify the ftp.rules file. The new rule will capture all traffic to and from the external host triggering the alert for a period of 300 seconds.

The Updated Snort Rule added to local.rules:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP command
overflow attempt – Tracked"; flow:to_server,established,no_stream;
dsize:>100; reference:bugtraq,4638; classtype:protocol-command-
decode; sid:1748; rev:4; tag: host, 300, seconds, src;)
```

As at 31 May 2004 the venerability still exist in the software as the vendor has not released on updated version[8]. It strongly suggested that an audit confirm no one within the organisation is using the vulnerable software.

3.10 Multiple Choice Question:

What is the primary purpose of sending a overly long and syntactically invalid command to a ftp server.?
(a) invalid commands simply get dumped by the ftp server.
(b) provided it is less than one datagram it wont be invalid.
(c) to simulate a response which might leak information about the server.
(d) the rfc959 is very flexible and as such it is ok to send long commands.

The best answer is (c).
(a) is wrong, the server should always return a 3 digit response code.
(b) is wrong, there is no limit to the size. Also remember at the application layer tcp is a stream and as such the server doesn't know how many datagrams were used.
(d) while being quite long it clearly outlines what is a valid command structure and exchange sequence.

Reference:

---

[1] Reining, Chris
LOGS: GIAC GCIA Version 3.4 Practical Detect Chris Reining, (8/2/2004)
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00064.html (28/5/2004)

[2] IEEE OUI and Company_id Assignments
URL: http://standards.ieee.org/regauth/oui/oui.txt (30/5/2004)

[3] Phaal, Peter.

Detection NAT Devices using sFlow
URL: http://www.sflow.org/detectNAT/ (19/06/2004)

[4] Msh, Skyrim.
3Com 3Cdaemon Buffer Overflow Vulnerability. (Apr 30, 2002)
URL: http://www.securityfocus.com/bid/4638/info/ (31/05/2004)

[5] Postel, J and Reynolds, J
File Transfer Protocol, Request For Comment 959 (October 1995)
URL: http://www.javvin.com/protocol/rfc959.pdf (2/5/2004)

[6] Sourcefire Research Team,
Snort Signature Database.
URL: http://www.snort.org/snort-db/sid.html?sid=1748 (30/5/2004)

[7] Jagielski, Jim.
Apache 1.3.27 Released (3/10/2002)
URL: http://archives.neohapsis.com/archives/apache/2002/0019.html (30/5/2004)

[8] 3Com Software Library – Additional Files – Utilities for Windows 32 Bit
URL: http://support.3com.com/software/utilities_for_windows_32_bit.htm (31/05/2004)

Question 3: Analysis This

1. Executive Summary

Effectively securing a single personal computer at home on an ADSL line can be a daunting task; securing a whole university is mammoth task. The additional challenges your IT staff face are:

- End users such as students and dare I say it lecturers may be a lot less cooperative than desired.
- Universities tend to be connected to the internet with fat pipes, this network resource is sought after by shady individuals wishing to wreak havoc.
- A lot of the students have no interest in computers but are at university to do a major other than IT.
- Some students with too much spare time tend to be creative.

In this report I review how effective your IT staff have been in maintaining the security of your infrastructure. This review specifically looks at the period Wednesday 7th April, 2004 through to and including Sunday 11th April, 2004. However I do believe the results obtained over this 5 day period really reflects how effective your policies, IT staff and Security specialist have been in the months leading up to the audit.

Over the 5 day period there were:

- 93, 379 alerts trending slightly up
- 15,683,216 port scans also trending slightly up
- 5,891 Out of Specification packet trending slightly down

Of the 93,379 alerts there were 51 different categories detected by your network based intrusion detection probe. In this report we look in depth at the 7 highest occurring alerts and the 10 internal hosts most actively scanning.

Overall, considering the size of your university network, it is quite secure. There are however a couple of issues which need your immediate attention:

- At least 18 hosts appear to have been compromised, 14 of which appear to be infected with a fast scanning worm.
- At least 12 of the previous 14 hosts appear to be 'bots' remotely controlled via IRC channels.
- One compromised host remotely controlled via VNC.
- Ingress filtering needs enhancing.
- DNS servers need to be dedicated to outgoing traffic.

2. Origin of the Logs

All logs were downloaded from http://www.incidents.org/logs/ as outlined in assignment requirement, however you will note that I'm just outside my 60-day limit. This was unavoidable as it was impossible to get anything near 5 continuous days at the end of April; even though the assignment suggested filling the gap with files immediately before the gap, in all cases the gap was excessive.

2.1 List of Files Analysed

Date	Alert File		Scan File		Out of Spec	
Apr 04	Name	Size	Name	Size	Name	Size
7	alert.040407.gz	1,714,968	scans.040407.gz	28,569,712	oos_report_040407 #2	3,456,000
8	alert.040408.gz	3,918,959	scans.040408.gz #1	8,937,472	oos_report_040408 #2	1,341,440
9	alert.040409.gz	4,134,815	scans.040409.gz #1	21,184,512	oos_report_040409 #1 #2	516,096
10	alert.040410.gz	5,008,515	scans.040410.gz	41,031,562	oos_report_040410 #2	1,638,400
11	alert.040411.gz	4,977,930	scans.040411.gz #1	27,385,856	oos_report_040411 #2	360,448

The files tagged with #1 were partially corrupted. By using zcat instead of gunzip I was able to extract records up to the corruption. The files were corrupted as follows. Scans.040408 terminated at 03:51:32. Scans.040409 terminated at 11:38:42. Scans.040411 terminated at 14:26:32. In all cases one would expect them to finish close to 23:59. The text file oos_report_040405 (e.g. data for 9th) contained some binary data near the end of it at 05:55:49.

The files tagged with #2 represent the file containing the actual data for the day required.

3. Traffic and Network Analysis

The one-hour summary graph below graphically indicates the amount of activity over the five-day period. Please be aware that the y-axis is logarithmic and as a result any spikes are indeed very big spikes.

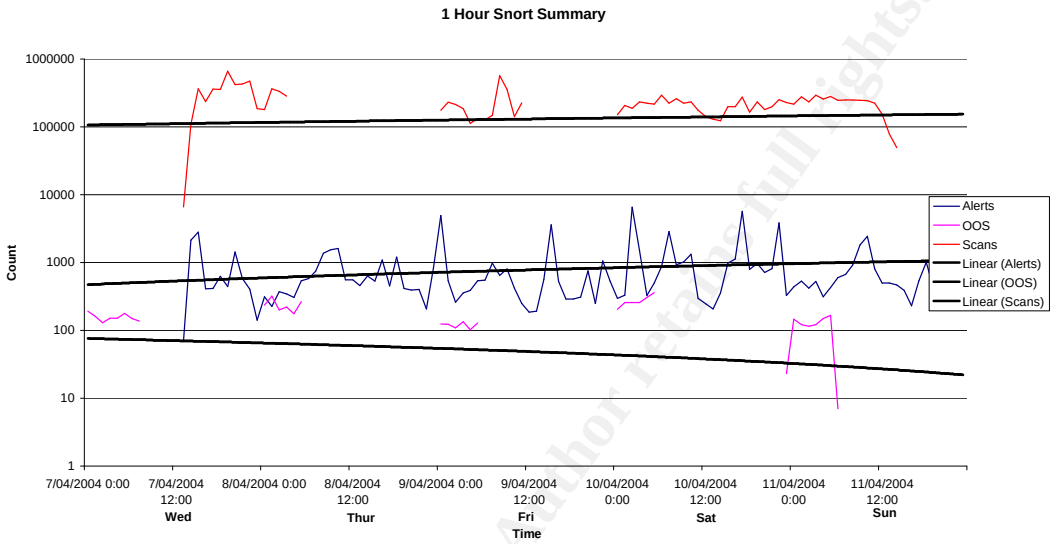
The first observation is that the logs appear broken. In the previous section I indicated that 4 of the downloaded files appeared corrupted toward the end, this is reflected in the graph.

Looking at the alerts we can see 8 distinct spikes in activity over the 5 days. The alert plot does appear to show a gap in data between 1:00am and 12:00 on the first day. Shown on the graph is an unfortunate trending up in the number of alerts recorded over the five-day period.

Next looking at the Out Of Spec (OOS) plot we see a very strange, yet very consistent, pattern. On all 5 days the pattern for OOS packets start just before midnight and abruptly stop just before 08:00am. Over the five- day period there is a

small trend down in the number of OOS packets but with such a fragmented plot the trend shouldn't be considered further.

Lastly looking at the scans plot uncovers some interesting patterns (ignoring the apparent break in data because of corrupted files). The first is that there were 3 distinct outbreaks of wide scale scanning; on one occasion spiking up to 660,000 scans in 1 hr. Also shown on the graph is an unfortunate trending up in the number of scans recorded over the five- day period.



3.1 Alerts

Alert Message	Total Alerts	Unique				Flow Direction			
		Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E
EXPLOIT x86 NOOP	28822	2035	1801			28822			
MY.NET.30.3 activity	12994	194	1			12994			
SMB Name Wildcard	12170			140	525		12170		
High port 65535 tcp - possible Red Worm – traffic	10664	75	46	48	105	5489	5175		
MY.NET.30.4 activity	10207	344	1			10207			
Tiny Fragments - Possible Hostile Activity	8005	10	14	1	1	8004	1		
DDOS mstream handler to client	3253			2	7		3253		
NMAP TCP ping!	1098	208	81			1098			
Possible trojan server activity	1081	23	271	18	44	849	232		
Null scan!	972	161	88			972			
External RPC call	930	2	260			930			
SUNRPC highport access!	637	29	24			637			
Incomplete Packet Fragments Discarded	511	100	81	1	1	510	1		
TCP SRC and DST outside network	309	29			90				309
High port 65535 udp - possible Red Worm – traffic	244	59	19	11	34	158	86		
ICMP SRC and DST outside network	210	50			207				210
[UMBC NIDS] Internal MiMail alert	158			3	102		158		

Alert Message	Total Alerts	Unique				Flow Direction			
		Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E
[UMBC NIDS IRC Alert] IRC user /kill detected	147	41	47	0	0	147			
DDOS shaft client to handler	142	23	1			142			
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	108			17	4		108		
FTP passwd attempt	100	91	1			100			
TCP SMTP Source Port traffic	83	3	1			83			
IRC evil - running XDCC	72			3	3		72		
EXPLOIT x86 setuid 0	66	55	35			66			
SMB C access	55	13	5			55			
[UMBC NIDS] External MiMail alert	47	19	1			47			
connect to 515 from outside	46	1	1			46			
EXPLOIT x86 setgid 0	33	26	27			33			
EXPLOIT x86 stealth noop	28	8	8			28			
[UMBC NIDS IRC Alert] Possible drone command detected.	25	3	8			25			
RFB - Possible WinVNC - 010708-1	24	5	6	6	5	13	11		
FTP DoS ftpd globbing	22	5	1			22			
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	17	3	2			17			
NIMDA - Attempt to execute cmd from campus host	15			9	3		15		
Attempted Sun RPC high port access	14	6	5			14			
TFTP - Internal UDP connection to external tftp server	14	6	2	3	6	7	7		
EXPLOIT NTPDX buffer overflow	10	8	7			10			
SYN-FIN scan!	9	9	9			9			
EXPLOIT x86 NOPS	8	1	1			8			
DDOS mstream client to handler	6	4	2			6			
Probable NMAP fingerprint attempt	5	5	5			5			
TFTP - External TCP connection to internal tftp server	4	2	2	1	1	3	1		
NETBIOS NT NULL session	3	1	3			3			
PHF attempt	2	2	1			2			
[UMBC NIDS IRC Alert] K\:line'd user detected	2	2	2			2			
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	2	1	1			2			
External FTP to HelpDesk MY.NET.53.29	1	1	1			1			
External FTP to HelpDesk MY.NET.70.49	1	1	1			1			
External FTP to HelpDesk MY.NET.70.50	1	1	1			1			
Fragmentation Overflow Attack	1	1	1			1			
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	1			1	1		1		
Totals:	93379	3665	2875	264	1139	71569	21291	0	519

Alerts Top 10 External talkers

External		Total scans	# Dst Hosts	Dst Port (Ports)
Source IP	**FQDN**			
212.76.225.24	cable-212.76.225.24.coditel.net	7559	2	16
199.131.21.34	zc7831522.ip.fs.fed.us	3480	500	3
68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	2993	1	2
141.157.102.155	pool-141-157-102-155.balt.east.verizon.net	2693	1	1
131.92.177.18	aeclt-cf00a4.apgea.army.mil	2166	1	1
68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	1660	2	1
69.138.77.62	pcp08479849pcs.desoto01.md.comcast.net	1628	2	1
68.43.170.140	bgp01087647bgs.waren301.mi.comcast.net	1566	87	3
68.55.113.194	pcp311543pcs.woodln01.md.comcast.net	1519	1	1
68.55.178.168	pcp233959pcs.elictc01.md.comcast.net	1298	2	1

Alerts Top 10 Internal Talkers

Internal Source IP	Total scans	Unique Dst Hosts	Unique Dst Ports (Dst' Port decreasing Frequency)
MY.NET.11.7	7016	2	1
MY.NET.84.235	3952	35	8
MY.NET.60.16	2169	2	2
MY.NET.111.228	991	1	1
MY.NET.150.198	674	167	1
MY.NET.150.44	632	165	1
MY.NET.97.51	618	1	1
MY.NET.75.13	598	171	1
MY.NET.11.6	524	2	1
MY.NET.97.92	379	1	1

Alerts Top 10 Alerts from External Hosts

Alert Message	Total
SMB Name Wildcard	12170
High port 65535 tcp - possible Red Worm - traffic	5175
DDOS mstream handler to client	3253
Possible trojan server activity	232
[UMBC NIDS] Internal MiMail alert	158
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	108
High port 65535 udp - possible Red Worm - traffic	86
IRC evil - running XDCC	72
NIMDA - Attempt to execute cmd from campus host	15
RFB - Possible WinVNC - 010708-1	11

Alerts Top 20 Alerts from Internal Hosts

Alert Message	Total
EXPLOIT x86 NOOP	28822
MY.NET.30.3 activity	12994
MY.NET.30.4 activity	10207
Tiny Fragments - Possible Hostile Activity	8004
High port 65535 tcp - possible Red Worm – traffic	5489
NMAP TCP ping!	1098
Null scan!	972
External RPC call	930
Possible trojan server activity	849
SUNRPC highport access!	637
Incomplete Packet Fragments Discarded	510
TCP SRC and DST outside network	309
ICMP SRC and DST outside network	210
High port 65535 udp - possible Red Worm – traffic	158
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	147
DDOS shaft client to handler	142
FTP passwd attempt	100
TCP SMTP Source Port traffic	83
EXPLOIT x86 setuid 0	66
SMB C access	55

Analysis of alerts with a count greater than 3000 over the 5 day period.

3.1.1 Alert #1 – EXPLOIT x86 NOOP

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
28822	2035	1801			28822				2004-04-07 00:08:16	04-11 23:46:32

Standard Snort SID's: none, however sid-648 [1] is a very close match.

The snort signature database description and GCIA practicals by Sai Prasad Kevavamatham [2] and Sylvain Randier [3] all indicate that this alert is normally a false positive.

dsport	Count
80	26032
1025	1753
5000	471
135	284
389	73

The first thing of note is that all alerts are for incoming traffic. Looking at the destination ports (see abridged table above) tells us a little more. First as expected port 80 is the highest. Initially I investigated the two highest sources and destinations of port 80 traffic, nothing was found unusual. I then decided to look a little deeper at the remaining ports as I believe it unusual to have such traffic incoming to a university. For example port 1025 is known to be used a back-door port as well as

having a number of vulnerabilities including one registered cve [4]. Likewise port 5000 is also commonly known as a back-door port. Port 389 (ldap) is used for authentication and also was targeted from one external source (216.65.73.26) to a number of internal destinations.

Recommendations:
Seriously consider whether ports such as 1025/5000/135/389 are required incoming. Unfortunately, even though this rule creates lots of false positives, it is valuable when trying to piece together how an incursion was perpetrated, thus you need to leave it enabled.

3.1.2 Alert #2 – MY.NET.30.3 activity

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
12994	194	1			12994				2004-04-07 13:13:16	04-11 23:38:53

Standard Snort SID's: none, custom alert for the university. It appears the university has implemented a custom alert to trigger on all incoming traffic to a Novell Web Enterprise Server.

The first thing of note is that all alerts are for incoming traffic. Looking at the destination ports (see abridged table below) tells us a little more. Port 524 is of particular interest due to its very high value. Novell's documentation [5] indicates that tcp 524 is used for Netware Core Protocol and udp 524 is used for NCP time synchronisation. From the alerts I'm unable to determine if it is tcp or udp. Even though the alerts span the whole review period I believe they were too erratic to be used for time synchronisation as such I believe they are tcp. A fellow GCIA student Peter Storm also indicated that this host was running other Netware protocols[6].

dsport	Count
524	12298
80	445
2745	75
6129	43
4899	20

Recommendations:
Consider whether tcp/udp port 524 needs to be available from the internet. Also consider splitting this catch all alert into three signatures and thus allow the alert message to indicate whether it is udp / tcp / other.

3.1.3 Alert #3 – SMB Name Wildcard

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
12170			140	525		12170			2004-04-07 00:11:04	04-11 23:39:49

Standard Snort SID's: none, custom alert for the university.
It appears the university has implemented a custom alert to trigger on all outgoing traffic with a destination port of udp137. While snort does not appear to have a matching rule it has appears to have been routinely used by various analyst. Bruce Alexander provided an interesting trace analysis using the same alert[7].

Originally, SMB name query traffic was always between two hosts both using udp 137 (see abridged table below). Samba, an open source implementation of SMB/CIFS is known to originate traffic from a source port other than 137[8]. An analysis of our traffic indicates that 140 of the hosts responsible for this traffic would appear to be Microsoft Windows work-stations. A further 2 work-stations (MY.NET.150.198, MY.NET.150.44) appear to behave both like a Samba and a Microsoft implementation.

srcport	Count
137	11041
1071	205
1050	199
1074	136
1051	124

Recommendations:
Identify why two work-stations act both like Samba and Microsoft. It may not be an issue e.g. DHCP is being used, but it would be easy to identify the primary use of such ip addresses. Consider whether you require outgoing udp port 137; in fact the university should consider the real benefits of allowing any SMB ports into and out of the campus.

3.1.4 Alert #4 – High port 65535 tcp – possible Red Worm - traffic

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
10664	75	46	48	105	5489	5175			2004-04-07 00:16:17	04-11 23:23:04

Standard Snort SID's: none, custom alert for the university.
It appears the university has implemented a custom alert to trigger any tcp traffic with either port of 65535.

Anthony Dell[9], as part of his GSEC certification, performed a very through analysis of how the Adore Worm worked. Anthony pointed out that the adore worm was originally known as Red Worm and propagated to any vulnerable Unix host.

Anthony and sequentially Peter Storm[6] indicated that any infected host would be expected to scan port 53, 111 and 515. Even though we had lots of flows to and from our network no hosts appear infected with the Adore worm.

```
Mysql Console:
mysql> create table ip_65535
    ->  select distinct srcip from alerts where message like
    ->    'High port 65535 tcp - possible Red Worm - traffic'
    ->    and srcip like 'MY.NET.%'
    ->  union distinct
    ->  select distinct dstip from alerts where
    ->    message like 'High port 65535 tcp - possible Red Worm - traffic'
    ->    and dstip like 'MY.NET.%';
Query OK, 51 rows affected (0.33 sec)
Records: 51  Duplicates: 0  Warnings: 0

mysql> create table ip_65535_2
    ->   select insert(srcip,1,6,'130.80') as srcip from ip_65535;
Query OK, 51 rows affected (0.01 sec)
Records: 51  Duplicates: 0  Warnings: 0

mysql> select distinct scans.srcip from scans,ip_65535_2 where
    ->   (scans.srcip=ip_65535_2.srcip) and
    ->   (scans.dstport=53 or scans.dstport=111 or scans.dstport=515);
Empty set (0.00 sec)
```

Recommendations:
This worm is quite old so effective patching as outlined by Anthony [9]should be very effective at preventing infection.

3.1.5 Alert #5 – MY.NET.30.4 activity

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
10207	344	1			10207				2004-04-07 00:08:22	04-11 23:45:10

Standard Snort SID's: none, custom alert for the university. It appears the university has implemented a custom alert to trigger on all incoming traffic to a Novell Web Enterprise Server.

The first thing of note is that all alerts are for incoming traffic. Looking at the destination ports (see abridged table below) tells us a little more. Port 51443 is of particular interest due to its very high value. Novell's documentation [10] indicates that tcp 51443 is used as a secondary https port when the Novell Web Enterprise Server is installed. We also see tcp/udp 524 as in alert 2 [3.1.2]. Once again it is impossible to determine if it is udp or tcp. Even though the alerts span the whole review period I believe they were too erratic to be used for time synchronisation as such I believe they are tcp. A fellow GCIA student Peter Storm also indicated that this host was running other Netware protocols[6].

Dsport	Count
51443	7255
80	2253
524	447
6129	59
4899	20

Recommendations:

Consider whether tcp/udp port 524 needs to be available from the internet. Also consider splitting this catch all alert into three signatures and thus allow the alert message to indicate whether it is udp / tcp / other.

3.1.6 Alert #6 – Tiny Fragments – Possible Hostile Activity

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
8005	10	14	1	1	8007	1			2004-04-07 17:32:05	04-11 08:58:02

Standard Snort SID's: none, custom alert for the university. The current version of snort contains a very similar alert [11]. Older versions of snort contained an minfrag preprocessor which was depreciated in Snort 1.8 in favour of stream4 preprocessor[12]. The associated documentation for using the old minfrag preprocessor directly matches the above alert [13].

Extract from Marty's 1.3 user documentation:

```
alert tcp any any -> any any (minfrag: 256; msg: "Tiny fragments detected,
possible hostile activity";)
```

Marty points out in his documentation[12] and in online discussions[14] that it is "unusual" for equipment to generate fragmented packets smaller than 256. Looking at the alerts it is impossible to determine either the size of the packets or the threshold. Of the recorded alerts some 94% were between MY.NET.43.3 and 212.76.225.24, additionally all but one of the alerts was incoming.

Below is a sample of the alerts between MY.NET.43.3 and 212.76.225.24.

Dttime	message	Srcip	srcport	Dstip	dstport
10/04/2004 0:30	Tiny Fragments - Possible Hostile Activity	212.76.225.24	0	MY.NET.43.3	0
10/04/2004 0:30	Tiny Fragments - Possible Hostile Activity	212.76.225.24	0	MY.NET.43.3	0
10/04/2004 0:30	Tiny Fragments - Possible Hostile Activity	212.76.225.24	0	MY.NET.43.3	0

As you can see neither the source port nor the destination port is set so it is very hard to determine which service is being used/targeted.

Looking at alerts, OOS and scans does not provide much more of a insight. The external host 212.76.225.24 (see registration information) only has alerts for "Tiny Fragments – Possible Hostile Activity" and "Null scan!", however no scans or OOS associated with it. The internal MY.NET.43.3 had 70 other alerts, none of immediate concern. There are 152 oos packets however they all appear to be standard ECN syn packets[6]. There were 4626 outgoing scans recorded against MY.NET.43.3; most were soulseek (tcp 2234) and some were edonkey(tcp 4662) [21]. While the scan preprocessor recorded them as scans I believe they were a result of heavy use of the respective P2P clients.

Sai Prasad Kesavamatham[2] also analysed similar alerts and believes it is a result of mutlicast traffic as part of the Access Grid Project. Unfortunately I do not have enough data to draw such a conclusion.

Recommendations:

Once again we see P2P heavily in use, as outlined elsewhere in this document the university should consider its use. To help identify exactly what traffic is causing this alert I suggest the university use the latest version of snort. I would further suggest they modify the rule (see highlighted below) to temporarily apply tagging. The rule is found in "misc.rules" file. You will also note that I have added a '+' to the frag bits, this is intentional as I have seen some GRE tunnels fragment packets with DF bit set. Interestingly they leave the DF bit set, thus you end up with fragments with DF bit set. I like it when vendors read rfc's.

```
Extract From misc.rules:
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments";
fragbits:M+; dsize: < 25; classtype:bad-unknown; sid:522; rev:1; tag:
host, 300, seconds, src; )
```

3.1.7 Alert #7 – DDOS mstream handler to client

Total Alerts	Unique				Flow Direction				Timestamp	
	Ext Src	Int Dst	Int Src	Ext Dst	In-bound	Out-bound	I->I	E->E	First	Last
3253			2	7		3253			2004-04-09 05:29:41	04-11 06:00:02

Standard Snort SID's: two, tcp 12754 has a snort signature of 248[15] and tcp 15104 has a snort signature of 250[16].

dstport	Count		srcport	Count
25	5		12754	3243
4662	3248		15104	10

From the above table it is apparent that both signatures are matching. All 5 alerts to dstport 25 were for host MY.NET.60.17, while there were a few alerts for this host nothing was extraordinary. As such I have ruled out the 5 alerts to destination port 25 as false positives.

The 3248 to port 4662 are of more concern. All of them originated from host MY.NET.84.235. Below is you can see that 3240 had a source address of MY.NET.84.235:12754 and a destination address of 82.48.242.184:4662.

srcip	dstip	dstport	srcport	Cnt
MY.NET.84.235	82.48.242.184	4662	12754	3240
MY.NET.60.17	65.54.252.99	25	15104	5
MY.NET.84.235	62.42.66.52	4662	12754	3
MY.NET.84.235	81.102.85.92	4662	15104	2
MY.NET.84.235	217.236.97.47	4662	15104	1
MY.NET.84.235	80.15.47.94	4662	15104	1
MY.NET.84.235	81.69.163.174	4662	15104	1

Taking a closer look at MY.NET.84.235 I was able to determine that it was very aggressively scanning for eDonkey clients[24]. Please refer to scan #9 latter in this document [3.2.9]. The first alert recorded between these two hosts was at 2004-04-10 22:19:55 and the last 2004-04-10 22:55:11. By looking at all scans originating from MY.NET.84.235 just before and after the initial alert I was able to confirm that source ports 12753 and 12755 were used to other random hosts. I was unable to confirm our outgoing stimulus to 82.48.242.184 but believe it must have been missed by the portscan preprocessor.

While Scan #9 confirms that MY.NET.84.235 was in fact scanning for eDonkey I had to explain why one scan, which coincidently was detected as 3240 “DDOS mstream handler to client” alerts. The first possible explanation was that a eDonkey client used within the university happened to use a source port of 12754 to log into 82.48.242.184 and download a file with lots of ‘>’ in it, possible but unlikely. The second, and most likely reason is a university host scanning for eDonkey clients locked up to host 82.48.242.184 for a period of 36 minutes. It is hard to say without knowing the type of scanner being used why MY.NET.84.235 locked up. Two possible reasons come too mind, first the scanner is crude and prone to lockup or secondly 82.48.242.184 may have been running a LaBrea[8] like tarpit for eDonkey scanners.

Recommendations:
As all these alerts appear to be false positives there is very little to do. However host MY.NET.84.235 does need remedial work, please see scan #9 [3.2.9] for further details.

3.2 Scans

Top Scan Types

Scantype	flags	count		Scantype	flags	Count
SYN	******S*	9219429		NOACK	**U*P*S*	41
UDP	0	6394838		NOACK	**U**RS*	37
FIN	*******F	58871		VECNA	**U*****	34
SYN	12****S*	7629		UNKNOWN	12***R**	30
NULL	********	493		UNKNOWN	1****R**	30
INVALIDACK	***A*R*F	477		UNKNOWN	*2*A**S*	26
UNKNOWN	*2***R**	60		UNKNOWN	1**A*R**	23
VECNA	****P***	58		NOACK	****P*S*	22
NOACK	**U**RSF	52		XMAS	*2U*P**F	19

Scans Top 10 External talkers

Srcip	Total scans	Unique Dst Hosts	Unique Dst Ports (Dst’ Port decreasing Frequency)
213.180.193.68	51559	1	51559 (random, none twice)
203.251.69.205	28392	15568	1(80)
61.146.52.26	28219	15567	1(80)
210.221.193.137	28189	15528	1(20168)
138.100.42.180	27798	15569	1(80)
24.97.20.62	27447	15515	1(4000)
194.79.163.149	27233	15405	1(554)
136.142.36.112	26338	15320	1(6129)
205.118.75.10	26166	15287	1(4000)
64.218.200.19	25657	15027	3(most 6129,32788, 32783)

Scans Top 10 Internal Talkers

Srcip	Total scans	Unique Dst Hosts	Unique Dst Ports (Dst’ Port decreasing Frequency)
MY.NET.1.3	2889682	103806	1474 (53, some 123,10123, .)

Srcip	Total scans	Unique Dst Hosts	Unique Dst Ports (Dst' Port decreasing Frequency)
MY.NET.111.51	1623396	1623037	112 (135, some 411, 13701..)
MY.NET.153.35	1522488	97068	6086 (1214, some 2067, ...)
MY.NET.81.39	1189494	1189003	15 (135, some 80, 10862 …)
MY.NET.70.96	1130689	89932	13 (#)
MY.NET.112.152	1082032	139714	15 (#)
MY.NET.1.4	796489	58554	695 (53, some 123, 45197 …)
MY.NET.66.56	338569	37158	13 (#)
MY.NET.84.235	295215	20414	1216 (4662, some 4661, ..)
MY.NET.42.2	253159	39941	12 (#)

very evenly distributed with ports 135,139, 445, 1025, 2745, 3127, 3410, 5000, 6129

Scan Analysis

Analysis of 10 hosts with the highest scan counts over the 5 day period.

3.2.1 Scan #1 – MY.NET.1.3

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
2889682	103806	1474	2004-04-07 00:08:05	2004-04-11 14:48:21

Below is a top-N profile of scans to/from MY.NET.1.3 grouped on port and scan type. Of greatest concern is outgoing "scans" to port 53 udp. The service most commonly found on udp 53 is dns.

Source MY.NET.1.3			Destination MY.NET.1.3	
Dstport	scantype	Count	srcport	Count
53	UDP	2877966	53	25
123	UDP	9019	32768	3
10123	UDP	325	32770	2
53	SYN	300	32774	2
45190	UDP	245	32938	2

Below is a sample of the first few scan records analysed based on time.

Dttime	srcip	srcport	Dstip	dstport	scantype	flags	info
7/04/2004 0:08	MY.NET.1.3	32783	128.193.0.30	53	UDP	0	UDP
7/04/2004 0:08	MY.NET.1.3	32783	209.208.0.96	53	UDP	0	UDP
7/04/2004 0:08	MY.NET.1.3	32783	216.127.43.91	53	UDP	0	UDP
7/04/2004 0:08	MY.NET.1.3	32783	217.160.72.252	53	UDP	0	UDP
7/04/2004 0:08	MY.NET.1.3	32783	65.198.177.5	53	UDP	0	UDP

From the above data outgoing DNS traffic represents 99.6% of all traffic. Additionally there is also a reasonable ratio (28:1) of total scans (o/g flows) to unique destination hosts; a ratio substantially higher than 1:1 would be expected of random internet browsing. I believe that the traffic above is valid traffic for a primary outgoing DNS server for this university.

A simple dns request (16/06/2004) for www.microsoft.com was resolved by the dns server, this confirms that the dns server is now acting as a external dns server. It is

unlikely that it has only just become a dual purpose dns server so I'm assuming it was a dual purpose DNS server during the period of this report.

Recommendations:
Ensure such a high use, high profile DNS server is either an internal or external DNS server but not both. A very good article by Joe Stewart explains the increased risks of cache poisoning [17] as a result of using the one dns server as both internal and external resolver.

3.2.2 Scan #2 – MY.NET.111.51

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
1623396	1623037	112	2004-04-07 00:08:08	2004-04-11 14:48:15

Below is a top-N profile of scans to/from MY.NET.11.51 grouped on port and scantype. Of greatest concern is outgoing "scans" to port 135 tcp. The service most commonly found on tcp 135 is epmap (Microsoft's DCE endpoint resolution)[18].

Source MY.NET.11.51			Destination MY.NET.11.51		
Dstport	Scantype	Count	Dstport	Scantype	Count
135	SYN	1622974	80	SYN	14
411	SYN	84	20168	SYN	8
13701	SYN	69	6129	SYN	6
1605	SYN	55	4899	SYN	3
413	UDP	19	1025	UDP	2

Below is a sample of the first few scan records analysed based on time.

Dttime	Srcip	srcport	Dstip	dstport	scantype	flags	Info
7/04/2004 0:08	MY.NET.111.51	4572	130.104.231.237	135	SYN	******S*	SYN ******S
7/04/2004 0:08	MY.NET.111.51	4573	130.104.231.238	135	SYN	******S*	SYN ******S
7/04/2004 0:08	MY.NET.111.51	4575	130.104.231.239	135	SYN	******S*	SYN ******S
7/04/2004 0:08	MY.NET.111.51	4576	130.104.231.240	135	SYN	******S*	SYN ******S

Below is a list of all alerts to / from MY.NET.111.51 over the 5 day period.

Dttime	Message	Srcip	srcport	Dstip	dstport
8/04/2004 13:05	EXPLOIT x86 NOOP	209.214.97.96	4352	MY.NET.111.51	1025
9/04/2004 23:39	RFB - Possible WinVNC - 010708-1	MY.NET.111.51	5900	68.55.192.251	62931
9/04/2004 23:39	RFB - Possible WinVNC - 010708-1	68.55.192.251	62931	MY.NET.111.51	5900
10/04/2004 21:44	RFB - Possible WinVNC - 010708-1	MY.NET.111.51	5900	68.55.192.251	62902
10/04/2004 21:44	RFB - Possible WinVNC – 010708-1	68.55.192.251	62902	MY.NET.111.51	5900
11/04/2004 2:53	RFB - Possible WinVNC – 010708-1	MY.NET.111.51	5900	68.55.192.251	60253
11/04/2004 2:53	RFB - Possible WinVNC – 010708-1	68.55.192.251	60253	MY.NET.111.51	5900
11/04/2004 6:56	EXPLOIT x86 setuid 0	217.215.120.150	3573	MY.NET.111.51	19992
11/04/2004 8:12	EXPLOIT x86 setuid 0	217.215.120.150	3573	MY.NET.111.51	19992
11/04/2004 18:57	RFB - Possible WinVNC – 010708-1	68.55.192.251	63445	MY.NET.111.51	5900

From the above data outgoing TCP-135 traffic represents 99.97% of all traffic. Additionally there is a ratio (1:1) of total scans (e.g o/g connection) to unique destination hosts; this gives further evidence to the fact this host is being used to identify potential targets. From the alerts above it would appear that it is being remotely controlled via VNC by an external host 68.55.192.251 (pcp229440pcs.catonv01.md.comcast.net). Additional detail about this host is contained in the registration section [6].

Recommendations:
Immediately isolate MY.NET.111.51 from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also look through all available logs (ids, firewall, syslog, windows event log, etc) to determine both the infection vector and timeline of the host's initial compromise. Finally you should notify the respective Org Abuse contact see [6]. You should also consider blocking known remote control ports e.g tcp-5900 on your firewall(s).

3.2.3 Scan #3 – MY.NET.153.35

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
1522488	97068	6086	2004-04-09 00:00:01	2004-04-11 12:47:06

Below is a top-N profile of scans to/from MY.NET.153.35 grouped on port and scan type. Of greatest concern is outgoing "scans" to port 1214 udp. The service most commonly found on udp 1214 is KazaA[19][20]. This port is also known to be used by other P2P programs such as Morpheus, Grokster and Fasttrack[21]. Note there also were 1058 outgoing connections to tcp port 1214.

Source MY.NET.153.35			Destination MY.NET.153.35		
Dstport	scantype	Count	dstport	scantype	Count
1214	UDP	19493	3247	SYN	102
2067	UDP	3930	3247	NOACK	21
3835	UDP	3595	3247	INVALIDACK	20
32656	UDP	3293	3247	NULL	16
2376	UDP	3184	80	SYN	11
3185	UDP	3043	3247	FIN	11
2695	UDP	2954	3247	VECNA	10

Below is a sample of some relevant scan records analysed based on time.

Dttime	srcip	srcport	dstip	dstport	scantype	flags	Info
9/04/2004 0:00	MY.NET.153.35	3247	12.223.234.133	3019	UDP	0	UDP
9/04/2004 0:00	MY.NET.153.35	3247	128.255.166.220	3653	UDP	0	UDP
9/04/2004 0:00	MY.NET.153.35	3247	24.91.130.4	2010	UDP	0	UDP
9/04/2004 0:00	MY.NET.153.35	3247	63.13.138.227	2389	UDP	0	UDP
cut							
9/04/2004 0:00	MY.NET.153.35	3351	162.40.171.84	1214	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.153.35	3247	172.128.81.21	1214	UDP	0	UDP

In this case 99.3% of outgoing traffic originates from one ephermal port 3247. Because KazzA is subsequently being used I believe this host is being used to identify 'live' internet hosts by initially scanning random obscure remote udp ports.

This is being followed up with attempted KazzA scans. I was able to very that at least 90 KazzA scans were proceeded by a random scan. The objective he is to find suitable 'open' hosts for leaching[23].

Recommendations:
While there is no evidence the MY.NET.153.35 has been compromised it should be immediately be isolated from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also look through all available logs (ids, firewall, syslog, windows event log, etc) to determine both the infection vector and timeline of the host's initial compromise.

3.2.4 Scan #4 – MY.NET.81.39

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
1189494	1189003	15	2004-04-09 07:11:52	2004-04-11 14:48:04

The analysis of top-N profile of scans to/from MY.NET.81.39 grouped on port and scantype was very similar to scan 2 found previously in this document [3.2.2]. Once again the greatest concern is outgoing "scans" to port 135 tcp. The service most commonly found on tcp 135 is epmap (Microsoft's DCE endpoint resolution)[18].

Source MY.NET.81.39			Destination MY.NET.81.39		
Dstport	Scantype	Count	Dstport	scantype	Count
135	SYN	1188912	80	SYN	13
80	SYN	412	6129	SYN	7
10862	UDP	49	20168	SYN	7
23890	UDP	25	4000	SYN	3
23498	UDP	24	3389	SYN	2

In this case outgoing TCP-135 traffic represents 99.99% of all traffic. Once again there is a ratio (1:1) of total scans (e.g o/g connection) to unique destination hosts; this gives further evidence to the fact this host is being used to identify potential targets. Below is a sample of the first few scan records based on time.

Dttime	Srcip	srcport	Dstip	dstport	scantype	flags	info
9/04/2004 7:11	MY.NET.81.39	1039	207.46.104.20	1863	SYN	******S*	SYN ******S*
9/04/2004 7:11	MY.NET.81.39	1064	108.28.156.1	135	SYN	******S*	SYN ******S*
9/04/2004 7:11	MY.NET.81.39	1065	108.28.156.2	135	SYN	******S*	SYN ******S*
9/04/2004 7:11	MY.NET.81.39	1066	108.28.156.3	135	SYN	******S*	SYN ******S*
9/04/2004 7:11	MY.NET.81.39	1067	108.28.156.4	135	SYN	******S*	SYN ******S*
9/04/2004 7:11	MY.NET.81.39	1068	108.28.156.5	135	SYN	******S*	SYN ******S*

Unfortunately the analysis of all the alerts, shown below, was fruitless. I could not identify any indications of a backdoor on the host. While there are a number of incoming scans to known backdoor/remote control ports none appeared anything more than normal random scans.

Dttime	Message	Srcip	srcport	dstip	dstport
8/04/2004 9:50	EXPLOIT x86 NOOP	210.202.16.129	3660	MY.NET.81.39	1025
8/04/2004 12:56	EXPLOIT x86 NOOP	130.160.146.172	4225	MY.NET.81.39	5000
9/04/2004 5:13	EXPLOIT x86 NOOP	140.251.88.83	4397	MY.NET.81.39	5000

Recommendations:

Immediately isolate MY.NET.81.39 from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also look through all available logs (ids, firewall, syslog, windows event log, etc) to determine both the infection vector and timeline of the host's initial compromise. You should also try to determine if and how this work-station is being controlled.

3.2.5 Scan #5 – MY.NET.70.96

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
1130689	89932	13	2004-04-07 13:13:06	2004-04-09 08:53:23

Below is a top-N profile of scans to/from MY.NET.70.96 grouped on port and scantype. Of greatest concern is the highly bunched scan to 10 well known destination ports (see highlight).

Source MY.NET.70.96			Destination MY.NET.70.96		
Dstport	Scantype	Count	srcport	scantype	Count
135	SYN	127985	80	SYN	11
2745	SYN	126799	6129	SYN	6
445	SYN	126462	20168	SYN	6
3127	SYN	126321	4899	SYN	4
139	SYN	125562	4000	SYN	2
1025	SYN	125525	2812	SYN	2
6129	SYN	124859	57	SYN	2
3410	SYN	124859	40	INVALIDACK	2
5000	SYN	124163	3042	INVALIDACK	2
1981	SYN	75	2361	INVALIDACK	2

Sample of scans.

Dttime	Srcip	srcport	Dstip	dstport	scantype	flags	Info
7/04/2004 13:13	MY.NET.70.96	1755	130.66.249.77	1025	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	1757	130.66.249.77	445	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2374	130.100.178.119	5000	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2375	130.172.96.196	2745	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2379	130.172.96.196	135	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2382	130.172.96.196	1025	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2384	130.172.96.196	445	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2386	130.172.96.196	3127	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2387	130.172.96.196	6129	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2388	130.172.96.196	139	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2389	130.172.96.196	3410	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2390	130.172.96.196	5000	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2391	130.163.15.105	2745	SYN	******S*	SYN ******S*
7/04/2004 13:13	MY.NET.70.96	2392	130.163.15.105	135	SYN	******S*	SYN ******S*

Relevant alerts.

Dttime	Message	srcip	srcport	dstip	dstport
7/04/2004 15:32 * 4 times	[UMBC NIDS IRC Alert] Possible drone command detected.	128.122.66.204	7000	MY.NET.70.96	3221
7/04/2004 16:27	EXPLOIT x86 NOOP	130.235.188.219	1740	MY.NET.70.96	1025

Dttime	Message	srcip	srcport	dstip	dstport
8/04/2004 1:50	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	MY.NET.70.96	3726	128.122.66.204	7000
8/04/2004 7:38 * 2 times	[UMBC NIDS IRC Alert] Possible drone command detected.	128.122.66.204	7000	MY.NET.70.96	3726
8/04/2004 16:18	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	128.122.66.204	7000	MY.NET.70.96	3726
9/04/2004 14:39	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	128.122.66.204	7000	MY.NET.70.96	1044
9/04/2004 18:38	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	MY.NET.70.96	1031	128.122.66.204	7000
9/04/2004 22:27	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	MY.NET.70.96	1041	128.122.66.204	7000
11/04/2004 15:23	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	MY.NET.70.96	1102	146.151.53.178	7000

From the above data it would appear that the above bot is scanning for potential targets. It is also appears to be controlled via IRC channel hosted by 128.122.66.204 (KAPTEREV.ICAS.FAS.NYU.EDU.). Additional detail about this host is contained in the registration section [6].

Elsewhere in this report you will see three other hosts with the same profile. As detailed in SANS handler's Diary during April it is most likely this pattern is a result of variant of {Phat|Ago|Gao}bot [22]. I decided to determine if I could identify others hosts / 'bots' with the same profile which didn't reach the top 10. By using the following sql commands I able to predict with high certainly that the following hosts have the same bot. The IRC flag indicates whether it probably is being controlled by 128.122.66.204.

Host	IRC	Host	IRC	Host	IRC
MY.NET.42.2	✓	MY.NET.43.5		MY.NET.43.10	✓
MY.NET.66.56	✓	MY.NET.70.96	✓	MY.NET.80.5	✓
MY.NET.80.28	✓	MY.NET.80.224	✓	MY.NET.112.152	✓
MY.NET.150.199	✓	MY.NET.150.210		MY.NET.151.75	✓
MY.NET.153.174	✓	MY.NET.153.195	✓		

Mysql Console:

```
mysql> create table bad select srcip,dstport, count(*) as count from scans
where (dstport = 135 or dstport = 2745 or dstport = 445 or dstport = 3127
or dstport = 139 or dstport = 1025 or dstport = 6129 or   dstport = 3410
or dstport = 5000) group by srcip,dstport;

mysql> select srcip,count(*) as cnt  from bad where srcip like '130.85.%'
and count > 10 group by srcip having cnt >7;
```

Recommendations:
Immediately isolate MY.NET.70.96 (and others) from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also

look through all available logs (ids, firewall, syslog, windows event log, etc) to determine both the infection vector and timeline of the host's initial compromise. Read the article by TonkiGin about how evasive bot's can be when coupled with IRC[23]. Finally by using information contained within TonkiGin's article you may be able to identify the operators of the channel(s).

3.2.6 Scan #6 – MY.NET.112.152

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
1082032	139714	15	2004-04-07 15:43:02	2004-04-07 22:40:51

Port profile was very similar to scan 5 [3.2.5]. There were 111 alerts similar to those seen in scan 5. Once again 128.122.66.204 was the controller. Data has not been repeated but the same reduction techniques were used. Recommendations are the same as in scan 5.

3.2.7 Scan #7 – MY.NET.1.4

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
796489	58554	695	2004-04-07 00:08:07	2004-04-11 14:48:20

Port profile was very similar to scan 1 [3.2.1]. There were 85 alerts similar to those seen in scan 1. Data has not been repeated but the same reduction techniques were used. Recommendations are the same as in scan 1.

From the analysed data outgoing DNS traffic represent 98.8% of all traffic. Additionally there is also a reasonable ratio (14:1) of total scans (o/g packet) to unique destination hosts; a ratio substantially higher than 1:1 would be expected of random internet browsing. I believe that the traffic above is valid traffic for a secondary outgoing DNS server for this university.

A simple dns request (16/06/2004) for www.microsoft.com was resolved by the dns server, this confirms that the dns server is now acting as an external dns server. It is unlikely that it has only just become a dual purpose dns server so I'm assuming it was a dual purpose DNS server during the period of this report.

Recommendations:
Ensure such a high use, high profile DNS server is either an internal or external DNS server but not both. A very good article by Joe Stewart explains the increased risks of cache poisoning [17] as a result of using the one dns server as both internal and external resolver.

3.2.8 Scan #8 – MY.NET.66.56

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
338569	37158	13	2004-04-09 08:19:45	2004-04-09 11:49:44

Port profile was very similar to scan 5 [3.2.5]. There were 3 alerts similar to those seen in scan 5. Once again 128.122.66.204 was the controller. Data has not been

repeated but the same reduction techniques were used. Recommendations are the same as outlined in scan 5.

3.2.9 Scan #9 – MY.NET.84.235

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
295215	20414	1216	2004-04-09 00:09:43	2004-04-11 14:48:21

Below is a top-N profile of scans to/from MY.NET.84.235 grouped on port and scan type. Of greatest concern is outgoing "scans" to port 4662 tcp. The service most commonly found on tcp 4662 is eDonkey[24]. This port is also known to be used by other P2P programs such as Overnet[24]. You will also notice that there is considerable scans to port 4661,4665 which are also used by eDonkey.

Source MY.NET.84.235			Destination MY.NET.84.235		
Dstport	scantype	Count	Dstcport	scantype	Count
4662	SYN	216907	2745	SYN	49
4661	SYN	10539	4662	SYN	37
5662	SYN	4207	80	SYN	26
4246	UDP	3381	1025	SYN	15
4665	UDP	3375	0	NULL	12
80	SYN	2101	20168	SYN	5
4663	SYN	1727	6129	SYN	4
4660	SYN	1008	4000	SYN	4
4242	SYN	989	0	NOACK	3
2842	SYN	823	21	SYN	2

Below is a sample of the first few scan records analysed based on time.

Dttime	srcip	srcport	dstip	dstport	scantype	Flags	info
9/04/2004 0:09	MY.NET.84.235	18528	81.203.204.143	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18538	65.92.213.55	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18539	213.102.234.6	31509	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18540	80.59.160.145	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18541	81.33.210.224	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18542	217.127.98.176	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18543	80.36.108.187	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18519	217.125.191.213	4662	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18520	81.33.241.106	4661	SYN	******S*	SYN ******S*
9/04/2004 0:09	MY.NET.84.235	18547	218.17.215.189	4661	SYN	******S*	SYN ******S*

There were some 5550 alerts over the 5 day period record against this host.

Message	srcip	cnt
DDOS mstream handler to client	MY.NET.84.235	3248
EXPLOIT x86 NOOP	199.131.21.34	791
High port 65535 tcp - possible Red Worm - traffic	MY.NET.84.235	660
High port 65535 tcp - possible Red Worm - traffic	81.203.197.37	289
High port 65535 tcp - possible Red Worm - traffic	217.95.183.166	225
DDOS shaft client to handler	207.68.172.236	49
Possible trojan server activity	MY.NET.84.235	43
Null scan!	219.137.39.207	23
High port 65535 tcp - possible Red Worm - traffic	217.95.189.202	23
EXPLOIT x86 NOOP	199.131.21.37	19

Message	srcip	cnt
High port 65535 tcp - possible Red Worm - traffic	80.178.191.31	15
EXPLOIT x86 NOOP	203.186.80.19	14
DDOS shaft client to handler	80.33.84.164	12
Cut		
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	MY.NET.84.235	1

Initially I investigated the 3248 “DDOS mstream handler to client” alerts to see if they were valid. I was able to establish that they were in fact false positives as a result of the university host randomly selecting and the using out going ephemeral ports, which just happen to be ‘on occasions’ to be the same port typically used by the mstream DDOS handler. Please refer to Alert #7 analysis for additional correlations[3.1.7]. Like wise I was able to rule out the “DDOS shaft client to handler” alerts.

We previously have seen in this paper that both the “Red Worm [3.1.4]” and the “Exploit x86 [3.1.1]” alerts are both prone to false positives. By using a process of elimination I was able to determine, with reasonable certainly, that this host is really a bot under control of a IRC channel. The IRC channel is hosted on 131.96.118.15 , further details available in the registration section[6].

Recommendations:
Immediately isolate MY.NET.84.235 from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also look through all available logs (ids, firewall, syslog, windows event log, etc) to determine both the infection vector and timeline of the host’s initial compromise. Finally you should notify the respective Org Abuse contact see [6].

3.2.10 Scan #10 – MY.NET.42.2

Total Scans	Unique		Timestamp	
	Dst ip	Dst Port	First	Last
253159	39941	12	2004-04-09 08:30:43	2004-04-09 09:50:22

Port profile was very similar to scan 5 [3.2.5]. There were 5 alerts similar to those seen in scan 5. Once again 128.122.66.204 was the controller. Data has not been repeated but the same reduction techniques were used. Recommendations are the same as outlined in scan 5.

3.3 OOS

Snort generates out-of-spec (OOS) log entries whenever it detects packets with tcp flags outside what is expected e.g. what is defined in various rfc’s.

Tcp_flags	Count
12****S*	5638
********	112
****P***	61
12***R**	16
*****RSF	4

Over the 5 day period 5891 logs were recorded. See (above) an abridged summary table grouped on tcp flag combinations which triggered the logs. The pattern "12****S*" which has the highest hit ratio is commonly seen today as a result of new tcp stacks trying to negotiate Explicit Congestion Notification (ECN), which is explained in detail in rfc2481 [25]. A quick review indicates with reasonable certainly most if not all of these are false positives.

The next most recorded OOS pattern over 5 day review period had tcp flags of '********'. Interestingly 86 of the packets were between 68.121.194.43:ephermal -> MY.NET.12.4:110. The first was recorded 2004-04-07 00:27:12 and the last at 2004-04-11 05:48:02. On closer inspection I noticed a further pattern all packets had the same ip_id (highlighted in red below), this for normal IP traffic is extremely unlikely and as such I believe these packets are crafted.

```
Sample Packets:
04/07-00:27:12.380674 68.121.194.43:4870 -> MY.NET.12.4:110
TCP TTL:78 TOS:0x0 ID:4660 IpLen:20 DgmLen:40
******** Seq: 0x5A6A001  Ack: 0x28EA489E  Win: 0x800  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
04/08-03:44:10.737522 68.121.194.43:24070 -> MY.NET.12.4:110
TCP TTL:78 TOS:0x0 ID:4660 IpLen:20 DgmLen:40
******** Seq: 0x38B2001  Ack: 0x4E826A04  Win: 0x800  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Additionally there were 183 "Null Scan!" from 68.121.194.43 to MY.NET.12.4 over the period but at different times to when the OOS packet were recorded.

Recommendations:
Immediately isolate MY.NET.12.4 from your network. Do a through investigation of the host to ensure virus patterns etc are up to date. You should also look through all available logs (ids, firewall, syslog, windows event log, etc) to determine if you can identify any other issues with this host.

4. Top Priority Issues

P2P: There is evidence of P2P programs being used see [3.1.6] and [3.2.3] While it is not against the law to use such programs it is commonly known that they are used to exchange pirated music and software. The university should ensure they have suitable policies outlining conditions of using P2P software on university infrastructure. They should also ensure every user is aware of, and accept the conditions in writing prior to being allowed to use the university infrastructure.

DNS: It appears that 2 of the universities outgoing dns servers are working as split dns servers see [3.1.2] and [3.1.5. It is suggested that the university consider operating dns as either outgoing or incoming to minimise the likelihood of dns cache poisoning.

Remote Control Software: There is evidence of remote control software being used to control compromised hosts internal to the university[3.2.2] and with reasonable certainty [3.2.4]. These hosts need to be taken out of service and either rebuilt or suitably repaired. In addition the university needs to consider the benefits of allowing incoming commonly known remote control connections.

Compromised Hosts: There is evidence of a number of hosts being used as distributed scanners. There is further evidence that most of these hosts are also being controlled via IRC channels. These hosts need to be isolated and either re-built or suitably repaired. Further details are found in the section “Insights on Internal Machine” [8].

5. Top Talkers

Please refer to following previous tables:

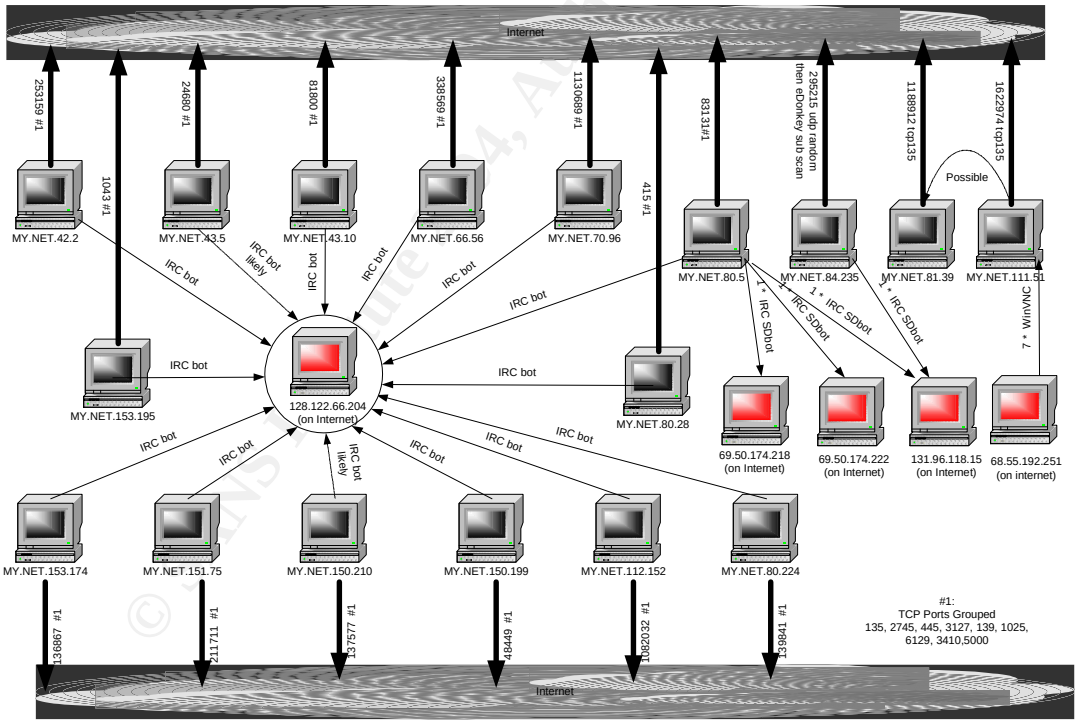
- Alerts Top 10 External Talkers
- Alerts Top 10 Internal Talkers
- Scans Top 10 External Talkers
- Scans Top 10 Internal Talkers

6. Registration Information

Host / Reason	Registration Information	Contact Information
212.76.225.24 Alert #6 See 3.1.6	**Net Range:** 212.76.225.0 - 212.76.225.255 **Name:** CODITEL **Country:** BE	**Abuse:** abuse@coditel.be **Contact:** Yves Beckers **Phone:** +32 2 226 54 23 **fax-no:** +32 2 219 77 25 **e-mail:** yves.beckers@coditel.be
82.48.242.184 Alert #7 See 3.1.7 Scan #9 See 3.2.9	**Net Range:** 82.48.240.0 - 82.48.255.255 **Name:** TELECOM-ADSL-3 Telecom Italia S.p.A. **Country:** IT	**Abuse:** abuse@telecomitalia.it **Phone:** +39 06 36881 **e-mail:** ripe-staff@telecomitalia.it
68.55.192.251 Scan #2 See 3.2.2	**Net Range:** 68.55.0.0 - 68.55.255.255 **Name:** Comcast Cable Communications, Inc. **Address:** 3 Executive Campus 5th Floor **City**: Cherry Hill **StateProv:** NJ **Country:** US	**OrgAbuseEmail**: abuse@comcast.ne **TechPhone**: +1-856-317-7200 **TechEmail:** cips_ip-registration@cable.comcast.com **OrgAbusePhone:** +1-856-317-7272
128.122.66.204	**Net Range:**	**Contact:**

Host / Reason	Registration Information	Contact Information
Scan #5 See 3.2.5	128.122.0.0 - 128.122.255.255 Name: New York University **Address:** Academic Computing Facility 251 Mercer Street **City:** New York **StateProv:** NY **Country:** US	RUSSELL@NYU.EDU **TechPhone**: +1-212-998-3431
131.96.118.15 Scan #8 See 3.2.8	**Net Range:** 131.96.0.0 - 131.96.255.255 **Name:** Georgia State University **Address:** University Computer Center University Plaza **City:** Atlanta **StateProv:** GA **Country:** US	**TechName:** Heidt, Sam B. **TechPhone:** +1-404-651-4567 **TechEmail:** sheidt@gsu.edu

7. Link Graph



8. Insights on Internal Machines

The following hosts appear to have been compromised and need further immediate investigation.

HOST	Section	Why

HOST	Section	Why
MY.NET.42.2	3.2.5,3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.43.5	3.2.5	High Volume scanner to 10 well know ports
MY.NET.43.10	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.66.56	3.2.5,3.2.8	High Volume 'bot' scanner to 10 well know ports
MY.NET.70.96	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.80.5	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.80.28	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.80.224	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.81.35	3.2.4	host scanning for Microsoft DCE
MY.NET.84.235	3.1.7,3.2.9	'bot' Scanning for eDonkey clients
MY.NET.112.152	3.2.5,3.2.6	High Volume 'bot' scanner to 10 well know ports
MY.NET.111.51	3.2.2	VNC controlled host scanning for Microsoft DCE
MY.NET.150.199	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.150.210	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.151.75	3.2.5	High Volume scanner to 10 well know ports
MY.NET.153.35	3.2.3	Host scanning for KazaA clients
MY.NET.153.174	3.2.5	High Volume 'bot' scanner to 10 well know ports
MY.NET.153.195	3.2.5	High Volume 'bot' scanner to 10 well know ports

9. Defensive Recommendations

Throughout this document when a particular issue was identified I made recommendations those recommendations should also be considered with the following general recommendations.

Ingress / Egress filtering: There were 519 alerts generated for flows external to external. There are two main subgroups of alerts with external / external flows. The first group was rfc1918 addresses [26], depending on network topology and probe location it is ok to have these addresses within your network. The other group were clearly outside addresses. It is suggested the university be a good net citizen and revise both their ingress and egress filtering detailed in rfc2827[27].

Compromised Hosts: There are a number of compromised hosts. While anti-virus software won't prevent all infestations a fully managed centralised anti-virus solution can be very effective at minimising outbreaks. This also needs to be supplemented with understandable end-user training on how each user can help minimise the chance that they become the next victim.

10. Analysis Process

The analysis process could be broken down into the following major phases.

- Downloaded and read honours papers
- Downloaded respective log files from incidents.org
- Verified date and data ranges contained within downloaded files
    i. pcregrep, vi, head, tail
- Downloaded past students scripts, tweaked and got them running how I needed them to
    i. Les Gordens'[28] sum_alerts.pl and create_gciadb.sql

ii. Samuel Adams[29] parseAlerts.pl, parseScans.pl, parseOOS.pl

- Concatenated each of the relevant logs files into 3 respective files
- In the case of alerts and scans files
  i. Sorted based on date e.g sort –n ….
  ii. Removed any lines for invalid dates e.g. pcregrep
- Modified Les's[28] create_gciadb.sql, created database in mysql
- Loaded data into database with scripts based on Samel's[29] scripts
- Created alerts summary using Les's sum_alerts.pl
- Created various summary table's using manual sql statements based on headings used by Peter Storm[6]
- Then by iteratively using the following tools built up the report
  i. Mysql queries
  ii. Microsoft Excel 2000
  iii. Microsoft Word 2000
  iv. vi
  v. PFE32 (programmers file editor)
  vi. pcregrep
  vii. sort, uniq, head, tail
  viii. snort distribution documentation
  ix. Google

Lessons learnt:
Ensure you set aside suitable time to get scripts working including time to tweak scripts developed by others. I would strongly suggest anyone considering doing this certification do a dry run at creating a dummy report prior to enrolling, that way all your tools will be sharp and clean.

I also used Mysql 4.0, which unfortunately does not support sub-queries. I strongly suggest that either Mysql 4.1 be used or an alternative database which supports sub-queries. My SQL was a little rusty so I took a while to get up to speed; once again a dry run would have addressed this issue.

I also suggest you keep a journal of major event's on the internet around the time of the expected analysis period e.g. while you're are doing section 1 and 2 of this assignment the university that your about to report on is silently being hacked. If you have a journal of major virus, worm and vulnerability releases during that period it makes it easier to analyse the data.

Start and take small steps it eventually falls into place. Ensure you jot down notes, it helps later. Additionally, create separate log files for each alert, scan and oos analysed, that way your crunched data is in manageable chunks.

Ensure you have heaps of spare disk space on your souped up analysis box.

References Q3

[1] Hart, Jon and Houghton, Nigel.
Snort Signature Database,
SID 648, SHELLCODE x86 NOOP
URL: http://www.snort.org/snort-db/sid.html?sid=648 (20/06/2004)

[2] Kesavamatham, Sai Prasad.
Intrusion Detection and Analysis (7th July,2003)
URL: http://www.giac.org/practical/GCIA/SaiPrasad_Kesavamatham_GCIA.pdf (15/05/2004)

[3] Randier, Sylvain.
GCIA Practical Assignment
URL: http://www.giac.org/practical/GCIA/Sylvain_Randier_GCIA.pdf (15/05/2004)

[4] CAN-2003-0533
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533 (20/06/2004)

[5] Technical Information Document,
Ports and Protocols used by Netware 5.x and 6.x – TID10013531 (18th Jun 2003)
URL: http://support.novell.com/cgi-bin/search/searchtid.cgi?/10013531.htm (18/06/2004)

[6] Storm, Peter H.
GIAC Certified Intrusion Analyst (GCIA), Prctical Assignment (Nov 15, 2003)
URL: http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf (06/06/2004)

[7] Alexander, Bruce.
Practical Assignment for GCIA Certification, Detect 1 (June 7,2000)
URL: http://www.giac.org/practical/Bryce_Alexander.doc (20/06/2004)

[8] SANS Institute,
IDS Signatures and Analysis, Part 1 & 2, Ch 10 pg 24, Ch 5 pg 38 (Sydney 2004)

[9] Dell, Anthony D.
Adore Worm – Another Mutation (6th April 2001)
URL: http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf (20/06/2004)

[10] Technical Information Document,
What are the default common ports for Netware 6? – TID10071836 (31st Jan, 2003)
URL: http://support.novell.com/cgi-bin/search/searchtid.cgi?/10071836.htm (18/06/2004)

[11] Houghton, Nigel and Black, Nick.
Snort Signature Database,
SID 522, MISC Tiny Fragments
URL: http://www.snort.org/snort-db/sid.html?sid=522 (20/06/2004)

[12] Roesch, Martin.
Snort Users Manual, Snort Release: 1.8, (9thJuly, 2001)
URL: http://www.selso.com/doc/SnortUsersManual.pdf (21/06/2004)

[13] Roesch, Martin.
Writing Snort Rules, How to write snort rules and keep your sanity, Current as of version 1.3.1.2
URL: http://packetstormsecurity.nl/papers/IDS/snort_rules.htm

[14] Roesch, Martin
Tiny Fragments (May 14, 2000)
URL: http://archives.neohapsis.com/archives/snort/2000-05/0103.html (20/06/2004)

[15] Novak, Judy.
Snort Signature Database,
SID 248, DDOS mstream handler to client
URL: http://www.snort.org/snort-db/sid.html?sid=248 (12/06/2004)

[16] Novak, Judy.
Snort Signature Database,
SID 250, DDOS mstream handler to client
URL: http://www.snort.org/snort-db/sid.html?sid=250 (12/06/2004)

[17] Stewart, Joe
DNS Cache Poisoning – The Next Generation (27/01/2003)
URL: http://www.securityfocus.com/guest/17905 (18/06/2004)

[18] Dshield.org, Port Report
URL: http://www.dshield.org/port_report.php?port=135 (18/06/2004)

[19] Neohapsis Port List
URL: http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html (18/06/2004)

[20] Supernodes
URL: http://www.kazaa.com/us/help/faq/supernodes.htm#FAQ_supernodes_4 (14/06/2004)

[21] Joining P2P networks with MLDonkey through a Firewall,
URL: http://mldonkey.berlios.de/modules.php?name=Wiki&pagename=WhatFirewallPortsToOpen
(14/6/2004)

[22] Fendley, Scott
Handler's Diary April 18th 2004 (08/06/2004)
URL: http://isc.sans.org/diary.php?date=2004-04-18

[23] TonikGin.
XDCC – An .EDU Admin's Nightmare (Sept 11,2002)
http://www.cs.rochester.edu/%7Ebukys/host/tonikgin/EduHacking.html (19/06/2004)

[24] Joining P2P networks with MLDonkey through a Firewall,
URL: http://mldonkey.berlios.de/modules.php?name=Wiki&pagename=WhatFirewallPortsToOpen
(14/6/2004)

[25] Ramakrishnan, K and Floyd, S.
RFC2481 – A Proposal to add Explicit Congestion Notification (ECN) to IP (Jan 1999)
URL: http://www.faqs.org/rfcs/rfc2481.html (24/06/2004)

[26] Rekhter, Y and Moskowitz, B and Karrenberg, D and de Groot, G J and Lear, E
Address Allocation for Private Intranets (Feb 1996)
URL: http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1918.html (26/06/2004)

[27] Ferguson, P and Senie, D
Network Ingress Filtering (May 2000)
URL: http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc2827.html (26/06/2006)

[28] Gordon, Les M.
Intrusion Analysis – The Director's Cut! (Nov 22, 2002)
URL: http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc (03/03/2004)

[29] Adams, Samuel C.
Fun with Intrusion Detection (23 Jun 2003)
URL: http://www.giac.org/practical/GCIA/Samuel_Adams_GCIA.pdf (03/06/2004)