# Global Information Assurance Certification Paper

# GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 4.0

Steve Payne
SANS@HOME
July 2004

Date submitted: 12/26/04

# Table of Contents

# Abstract

This paper contains analysis of one day of Snort logs captured by a University. It is made up of three parts. In the first part, the executive summary, I will summarize my findings for those who may not have technical expertise in intrusion detection. The second part of the paper is the detailed analysis. This section describes my findings in a more technical manner. It contains an overview of the attacks made on the University network, and three of these attacks are given in-depth analysis. This section also contains statistics and diagrams based on the attack traffic, and some recommendations to the University on how to improve security. In the final section of the paper, I describe the process I used during the analysis done in the previous section.

## *Document Conventions*

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

| | |
|---|---|
| `Command` | Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell. |
| `filename` | Filenames, paths, and directory names are represented in this style. |
| `computer output` | The results of a command and other computer output are in this style. |
| URL | Web URL's are shown in this style. |
| *Quotation* | A citation or quotation from a book or web site is in this style. |

# Part I: Executive Summary

This report explains my findings after my analysis of your University's IDS log file for 10/31/2002. It also provides my recommendations on how to improve the state of security on the network.

While analyzing the alerts, I found many events that were of interest. The alerts caused by incoming traffic consisted of a wide range of attacks. There were scans looking for exploitable hosts, and attacks against one of the University's web servers. There were also a number of alerts that were false positives. These pose a problem because they take up analysts' time that they could be devoting to investigating real attacks.

There were a large number of alerts for traffic coming from the University network. Most of these had to do either with spyware or with Gnutella peer-to-peer file sharing. These two things need to be looked into due to both the number of alerts they are creating and the possible privacy and copyright issues.

There are many ways in which the overall security of the network can improve. Here are my recommendations for improving security. These recommendations, if followed, will help to insure a more secure network and a better intrusion detection system. More detail can be found in the "Defensive Recommendations" section found later in the report.

- Perform proper filtering at the network border. Many alerts have been caused by traffic that does not need to be allowed to enter the network. I recommend that all routers and firewalls be examined and stronger filtering be put in place.
- Tune the existing IDS to eliminate unwanted alerts and false positives.
- Place additional IDS on the internal NAT network in order to identify problem hosts.
- Inform users about spyware privacy issues and help them to eliminate spyware on their systems.
- Set up a system to capture all packets entering and leaving the network. This would help immensely when analyzing IDS events.
- Timely updating and patching of systems needs to be a priority. Many attacks are successful only because the target is running old software versions.
- If not currently being done, I recommend that a vulnerability scanner be used. Nessus (http://www.nessus.org/) is open-source and can be obtained at no cost to the University. This will allow possible threats to be eliminated before they become an issue.
- I recommend the use of host-based firewalls on all servers and workstations. This is an extra layer of defense that can prevent attacks that find a way to bypass your network firewalls.

# Part II: Detailed Analysis

## *1. Files Used for Analysis*

I chose to use the log file http://isc.sans.org/logs/Raw/2002.9.31 for my analysis. The file http://isc.sans.org/logs/Raw/README states the following regarding this file:

> *The log files are the result of a Snort instance running in binary logging mode. This means that only the packets that violate the ruleset will appear in the log.*

*The logs themselves have been sanitized. All of the IP addresses of the protected network space have been "munged". Additionally, the checksums have been modified to prevent clever people from discovering the original IP addresses. You will find that certain keywords within the packets have been replaced with "X"s. All ICMP, DNS, SMTP and Web traffic has also been removed.*

The data contained in the log file was collected on 10/31/2002 between 00:00:01 and 23:58:39 GMT.

## 2. Relationship Analysis

### 2.1 Network Topology

In order to analyze the captured packets more efficiently, it is helpful to know the structure of the network. No documentation was provided as to how the network was set up, but some information can be obtained from the data in the log file. I used tcpdump along with some other UNIX commands to produce the following information.

The first step I took was to examine the MAC addresses that the Snort sensor had logged.

I used the command `tcpdump -tennqr 2002.9.31 | cut -d " " -f 1-3 | sort -u` to get the following output:

```
00:00:0c:04:b2:33 > 00:03:e3:d9:26:c0,
00:03:e3:d9:26:c0 > 00:00:0c:04:b2:33,
```

This shows that there were only two distinct MAC addresses logged by Snort. I referenced these addresses against the IEEE public OUI listing located at http://standards.ieee.org/regauth/oui/oui.txt:

*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*

I will now refer to the device with MAC address 00:00:0c:04:b2:33 as “Cisco Device #1”. The device with MAC address 00:03:e3:d9:26:c0 will be referred to as “Cisco Device #2”.

Since the only MAC addresses seen were from Cisco devices, the Snort sensor is probably monitoring a network between two routers or firewalls. Due to the fact that the log only contains packets that generated alerts, there may be additional devices on this network whose traffic was not logged. The sensor could be receiving the data through a tap, span port, or hub.

After determining the location of the sensor, I wanted to find out where the University network was located. I examined the data with the command `tcpdump -tnnqr 2002.9.31 | less`. I noticed that every logged packet appeared to contain an address in the 207.166.0.0/16 network.
Output sample:

```
IP 207.166.87.157.63028 > 68.34.84.42.9859: tcp 330
IP 207.166.87.157.63029 > 208.20.78.214.8736: tcp 329
IP 148.64.138.224.3775 > 207.166.135.150.9511: tcp 118
IP 148.64.138.224.3793 > 207.166.135.150.9511: tcp 22
IP 148.64.138.224.3775 > 207.166.135.150.9511: tcp 118
IP 148.64.138.224.3793 > 207.166.135.150.9511: tcp 22
IP 216.77.216.150.57780 > 207.166.52.165.1080: tcp 0
```

To confirm that every packet contained traffic to or from the 207.166.0.0/16 network, I used the command `tcpdump -tnnqr 2002.9.31 'not net 207.166'`. This command returned no results. The 207.166.0.0/16 is most likely the University network.

I used the command `tcpdump -tnneqr 2002.9.31 'src net 207.166' | cut -d " " -f 1,8 | less` to find out the source MAC address for traffic originating from the University network.
Output sample:

```
00:00:0c:04:b2:33 207.166.87.157.61450
00:00:0c:04:b2:33 207.166.87.157.61456
00:00:0c:04:b2:33 207.166.87.157.61465
00:00:0c:04:b2:33 207.166.87.157.61465
00:00:0c:04:b2:33 207.166.87.157.61468
00:00:0c:04:b2:33 207.166.87.157.61469
00:00:0c:04:b2:33 207.166.87.157.61471
00:00:0c:04:b2:33 207.166.87.157.61468
00:00:0c:04:b2:33 207.166.87.157.61474
```

With the command `tcpdump -tnneqr 2002.9.31 'src net 207.166 and ether src 00:03:e3:d9:26:c0'`, I wanted to see if any traffic coming from the University network range had the source MAC address of Cisco Device #2. This command produced no results, so the University network is located behind Cisco Device #1. The external network, including the Internet, is located on the other side of Cisco Device #2.

While running previous commands, I had seen only one source address,

207.166.87.157, on the University network. I wanted to know if there were any others, so I ran the command `tcpdump -tnnqr 2002.9.31 'src net 207.166 and not host 207.166.87.157'`. The output showed me one additional source address:

```
IP 207.166.87.40.80 > 213.191.138.4.1148: tcp 536
IP 207.166.87.40.80 > 195.29.35.145.3126: tcp 485
IP 207.166.87.40.80 > 212.91.102.209.3027: tcp 563
IP 207.166.87.40.80 > 212.91.102.209.3058: tcp 563
IP 207.166.87.40.80 > 212.91.102.209.3139: tcp 563
```

The next step I took was to examine the University network source addresses in order to find out more about them. I was looking for information such as number of hops away from Cisco Device #1, host operating system, and what the host was being used for. I first looked at the traffic coming from 207.166.87.40.

The command `tcpdump -tnnvr 2002.9.31 'src host 207.166.87.40'` showed me some more verbose information about these packets.

```
IP (tos 0x0, ttl  63, id 35572, offset 0, flags [DF], proto 6, length: 576, bad cksum 727b
(->2831)!) 207.166.87.40.80 > 213.191.138.4.1148: P [bad tcp cksum 25ec (->dba1)!]
1897618253:1897618789(536) ack 3400321 win 32696
IP (tos 0x0, ttl  63, id 54166, offset 0, flags [DF], proto 6, length: 525, bad cksum a321
(->58d7)!) 207.166.87.40.80 > 195.29.35.145.3126: P [bad tcp cksum 5157 (->4730)!]
955738518:955739003(485) ack 3505970755 win 32120
IP (tos 0x0, ttl  63, id 53034, offset 0, flags [DF], proto 6, length: 603, bad cksum 52c1
(->877)!) 207.166.87.40.80 > 212.91.102.209.3027: P [bad tcp cksum 5189 (->2a7f)!]
2766936092:2766936655(563) ack 1936810831 win 32120
IP (tos 0x0, ttl  63, id 56721, offset 0, flags [DF], proto 6, length: 603, bad cksum 445a
(->fa0f)!) 207.166.87.40.80 > 212.91.102.209.3058: P [bad tcp cksum d99e (->b294)!]
3051434184:3051434747(563) ack 2004333420 win 32120
IP (tos 0x0, ttl  63, id 346, offset 0, flags [DF], proto 6, length: 603, bad cksum 2092 (-
>d647)!) 207.166.87.40.80 > 212.91.102.209.3139: P [bad tcp cksum 7b3d (->5433)!]
4036794895:4036795458(563) ack 2244760875 win 32120
```

The original TTL on these packets was probably 64. I referenced the Honeynet Project's fingerprinting traces at http://project.honeynet.org/papers/finger/traces.txt which provided the following match:

| *OS* | *VER* | *PLATFORM* | *TTL* | *WINDOW* | *DF* | *TOS* |
|---|---|---|---|---|---|---|
| *Linux* | *2.2.x* | *Intel* | *64* | *32120* | *y* | *0* |

I used the –X switch to output the packets in ASCII. `tcpdump -nnqXr 2002.9.31 'src host 207.166.87.40'`.

Output sample:

```
07:35:24.446507 IP 207.166.87.40.80 > 212.91.102.209.3139: tcp 563
        0x0000:  4500 025b 015a 4000 3f06 2092 cfa6 5728  E..[.Z@.?.....W(
        0x0010:  d45b 66d1 0050 0c43 f09c 9a0f 85cc 552b  .[f..P.C......U+
        0x0020:  5018 7d78 7b3d 0000 4854 5450 2f31 2e31  P.}x{=..HTTP/1.1
        0x0030:  2034 3033 2046 6f72 6269 6464 656e 0d0a  .403.Forbidden..
        0x0040:  4461 7465 3a20 5468 752c 2033 3120 4f63  Date:.Thu,.31.Oc
        0x0050:  7420 3230 3032 2031 383a 3235 3a32 3820  t.2002.18:25:28.
        0x0060:  474d 540d 0a53 6572 7665 723a 2041 7061  GMT..Server:.Apa
        0x0070:  6368 652f 312e 332e 3132 2028 556e 6978  che/1.3.12.(Unix
        0x0080:  2920 2028 5265 6420 4861 742f 4c69 6e75  )..(Red.Hat/Linu
        0x0090:  7829 2046 726f 6e74 5061 6765 2f34 2e30  x).FrontPage/4.0
```

The data contained in this packet appears to be a HTTP error which also includes the web server version (Apache/1.3.12) and the operating system (Red.Hat/Linux). The traffic originating from this address had a source port of 80.

Since the original TTL of 64 was only reduced by 1 when the packets were logged, and taking into account the port 80 traffic along with HTTP errors, this address probably belongs to a web server on a network that is directly connected to a different interface on Cisco Device #1.

I next examined the source IP address 207.166.87.157. The command `tcpdump -tnnvqr 2002.9.31 'src host 207.166.87.157' | less` provided some interesting information.

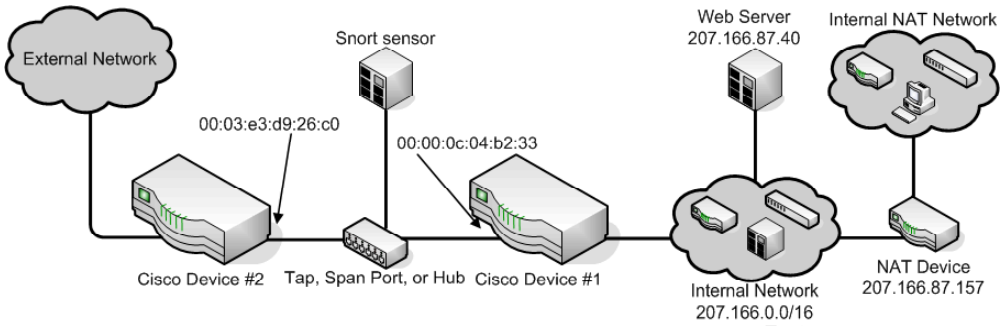
```
IP (tos 0x0, ttl 124, id 16811, offset 0, flags [DF], proto 6, length: 149, bad cksum 4ef5
(->4ab)!) 207.166.87.157.61054 > 64.154.8
0.47.80: tcp 109
IP (tos 0x10, ttl 240, id 0, offset 0, flags [none], proto 6, length: 3069, bad cksum 0 (-
>6de)!) 207.166.87.157.61054 > 64.154.80.4
7.80: tcp 3029
IP (tos 0x0, ttl 124, id 16830, offset 0, flags [DF], proto 6, length: 167, bad cksum 4ed0
(->486)!) 207.166.87.157.61055 > 64.154.8
0.47.80: tcp 127
IP (tos 0x10, ttl 240, id 0, offset 0, flags [none], proto 6, length: 1500, bad cksum 0 (-
>cff)!) 207.166.87.157.61055 > 64.154.80.4
7.80: tcp 1460
IP (tos 0x10, ttl 240, id 0, offset 0, flags [none], proto 6, length: 1627, bad cksum 0 (-
>c80)!) 207.166.87.157.61055 > 64.154.80.4
7.80: tcp 1587
IP (tos 0x0, ttl 124, id 23311, offset 0, flags [DF], proto 6, length: 369, bad cksum 74be
(->2a74)!) 207.166.87.157.61450 > 12.233.
67.215.9409: tcp 329
IP (tos 0x0, ttl 124, id 23409, offset 0, flags [DF], proto 6, length: 367, bad cksum f781
(->ad37)!) 207.166.87.157.61456 > 24.136.
181.20.8333: tcp 327
```

There are different TTL values, which could mean that 207.166.87.157 is a NAT device with multiple internal hosts using the same external IP address. I ran the command `tcpdump -tnnvr 2002.9.31 'src host 207.166.87.157' | awk '{print $4 " " $5}' | sort –u` to find out how many different TTL values existed:

```
ttl 123,
ttl 124,
ttl 125,
ttl 240,
ttl 29,
```

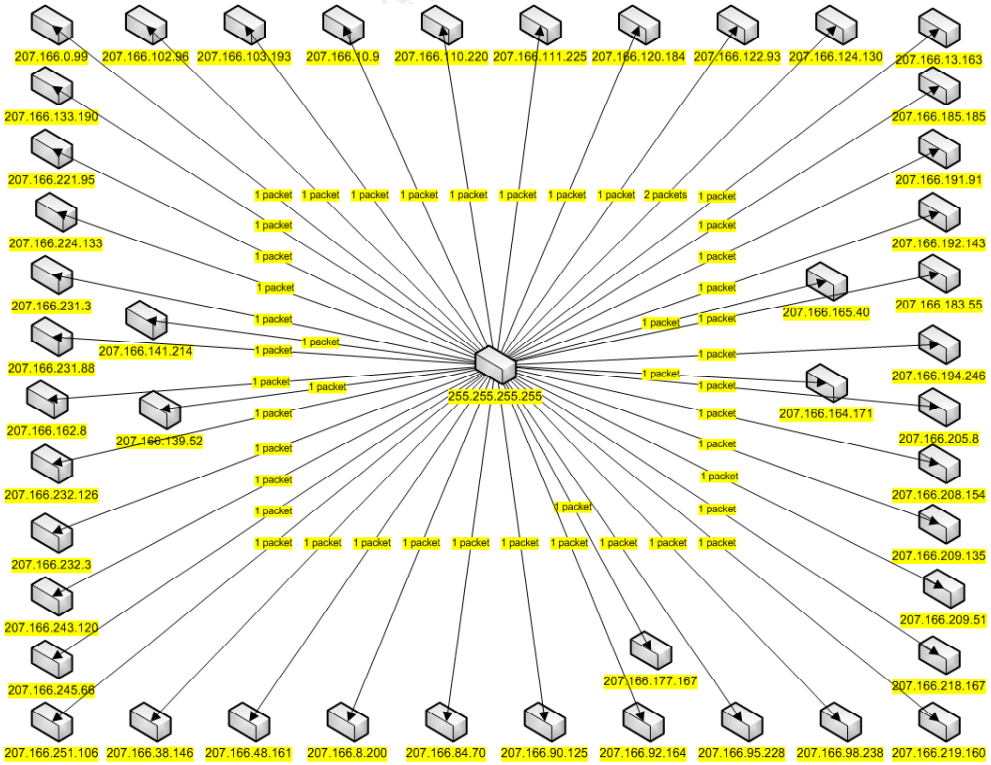
Comparing these TTL values to the Honeynet Project's fingerprinting traces at http://project.honeynet.org/papers/finger/traces.txt, it seems like the original TTL values would have been 128, 255, and 30 or 32. This means that there is a fairly complex network containing multiple routers behind the NAT address 207.166.87.157.

Not enough information has been provided to make an exact map of the network. Here is a diagram which illustrates one possible network layout:




## 2.2 Link Graph

This is a diagram of traffic with source IP address 255.255.255.255 that targeted 45 different IP addresses on the University network. Each address was sent one packet with the exception of 207.166.124.130, which was sent two. There were no packets recorded with a destination address of 255.255.255.255. Further examination of this traffic can be found in the “detects” section of this paper.



## 3. Detects

### Overview

I used BASE (http://base.secureideas.net/) to get the data and screenshots in this section. BASE is a web-based GUI for analyzing Snort alerts.

The total number of alerts generated by Snort was 4359. There were 32 unique alerts in 9 different categories. 70 source IP addresses and 1263 destination IP addresses were observed, making up 1327 unique IP links. All alerts were for TCP traffic, with 1569 source ports and 1056 destination ports. The following image shows the unique alerts generated by Snort. I have chosen three of these to report on.

| < Signature > | < Classification > | < Total # > | Sensor # | < Source Address > | < Dest. Address > |
|---|---|---|---|---|---|
| [snort] P2P Outbound GNUTella client request | policy-violation | 1614 (37%) | 1 | 18 | 1178 |
| [snort] P2P GNUTella client request | policy-violation | 1614 (37%) | 1 | 18 | 1178 |
| [arachNIDS] [snort] BACKDOOR Q access | misc-activity | 46 (1%) | 1 | 1 | 45 |
| [snort] (http_inspect) BARE BYTE UNICODE ENCODING | *unclassified* | 188 (4%) | 1 | 1 | 15 |
| [nessus] [nessus] [cve] [icat] [cve] [icat] [bugtraq] [bugtraq] [arachNIDS] [snort] WEB-CGI formmail arbitrary command execution attempt | web-application-attack | 6 (0%) | 1 | 3 | 1 |
| [nessus] [nessus] [cve] [icat] [cve] [icat] [bugtraq] [bugtraq] [arachNIDS] [snort] WEB-CGI formmail access | web-application-activity | 9 (0%) | 1 | 5 | 1 |
| [arachNIDS] [snort] SCAN FIN | attempted-recon | 2 (0%) | 1 | 2 | 1 |
| [snort] (snort_decoder) WARNING: TCP Data Offset is less than 5! | *unclassified* | 1 (0%) | 1 | 1 | 1 |
| [snort] SHELLCODE x86 NOOP | shellcode-detect | 13 (0%) | 1 | 2 | 1 |
| [nessus] [snort] WEB-FRONTPAGE _vti_inf.html access | web-application-activity | 13 (0%) | 1 | 3 | 1 |
| [nessus] [cve] [icat] [bugtraq] [nessus] [cve] [icat] [cve] [icat] [bugtraq] [bugtraq] [snort] WEB-FRONTPAGE shtml.exe access | web-application-activity | 13 (0%) | 1 | 3 | 1 |
| [nessus] [snort] WEB-FRONTPAGE /_vti_bin/ access | web-application-activity | 17 (0%) | 1 | 6 | 1 |
| [cve] [icat] [nessus] [bugtraq] [snort] WEB-FRONTPAGE _vti_rpc access | web-application-activity | 13 (0%) | 1 | 3 | 1 |
| [snort] (http_inspect) NON-RFC HTTP DELIMITER | *unclassified* | 4 (0%) | 1 | 2 | 3 |
| [cve] [icat] [cve] [icat] [cve] [icat] [bugtraq] [bugtraq] [bugtraq] [snort] WEB-MISC Chunked-Encoding transfer attempt | web-application-attack | 4 (0%) | 1 | 1 | 2 |
| [snort] ATTACK-RESPONSES 403 Forbidden | attempted-recon | 5 (0%) | 1 | 1 | 3 |
| [nessus] [bugtraq] [snort] WEB-MISC Invalid HTTP Version String | non-standard-protocol | 79 (2%) | 1 | 28 | 4 |
| [snort] BAD-TRAFFIC ip reserved bit set | misc-activity | 3 (0%) | 1 | 3 | 1 |
| [snort] CHAT IRC nick change | policy-violation | 7 (0%) | 1 | 1 | 1 |
| [snort] CHAT MSN message | policy-violation | 38 (1%) | 1 | 1 | 5 |
| [snort] (http_inspect) OVERSIZE REQUEST-URI DIRECTORY | *unclassified* | 398 (9%) | 1 | 1 | 8 |
| [snort] (http_inspect) DOUBLE DECODING ATTACK | *unclassified* | 7 (0%) | 1 | 1 | 5 |
| [snort] SHELLCODE x86 inc ebx NOOP | shellcode-detect | 2 (0%) | 1 | 2 | 1 |
| [snort] (http_inspect) IIS UNICODE CODEPOINT ENCODING | *unclassified* | 215 (5%) | 1 | 1 | 7 |
| [cve] [icat] [bugtraq] [snort] WEB-CGI redirect access | attempted-recon | 5 (0%) | 1 | 1 | 2 |
| [snort] WEB-ATTACKS cc command attempt | web-application-attack | 1 (0%) | 1 | 1 | 1 |
| [arachNIDS] [snort] SHELLCODE x86 NOOP | shellcode-detect | 14 (0%) | 1 | 2 | 1 |
| [cve] [icat] [bugtraq] [snort] WEB-IIS %2E-asp access | web-application-activity | 4 (0%) | 1 | 1 | 1 |
| [arachNIDS] [snort] SHELLCODE x86 setgid 0 | system-call-detect | 1 (0%) | 1 | 1 | 1 |
| [bugtraq] [arachNIDS] [snort] WEB-IIS view source via translate header | web-application-activity | 19 (0%) | 1 | 1 | 1 |
| [arachNIDS] [snort] WEB-MISC http directory traversal | attempted-recon | 2 (0%) | 1 | 1 | 1 |
| [arachNIDS] [snort] (http_inspect) WEBROOT DIRECTORY TRAVERSAL | attempted-recon | 2 (0%) | 1 | 1 | 1 |

## 3.1 Detect One - SHELLCODE x86 NOOP

### Description

An alert for SHELLCODE x86 NOOP may indicate a buffer overflow attack. To execute this kind of attack, the attacker will try to get an application running on the attacked host to put too much data into a buffer. The buffer will reach its limit, and any data that could not fit will go into memory overwriting what was previously there. This can possibly result in the host executing commands given to it by the attacker.

The rules which generate these alerts are disabled in the default Snort configuration with a note that enabling them will cause a large decrease in performance. If enabled, the default snort configuration causes them to inspect all traffic except that which is on port 80. Port 80 is probably left out due to the fact that many images or other binary files that are commonly downloaded over port 80 can contain strings of NOOP characters, which will generate false positives. As mentioned by Robert David Graham (http://archives.neohapsis.com/archives/sf/ids/2002-q2/0029.html), the traditional x86 NOOP code is 0x90, but others can be used including 0x61 which translates to an ASCII “a” character.

For more information on this type of attack, please refer to the article *“Smashing The Stack For Fun And Profit”* by Aleph One (http://www.phrack.org/show.php?p=49&a=14).

### Reason for Selection

I chose to examine this alert because a string of NOOP characters can indicate a buffer overflow attempt. Many buffer overflow attacks result in the attacker gaining root access to the attacked host. The target is a host behind the NAT address 207.166.87.157, which means that the attacker would have access to the internal network if the attack was successful.

### Detect Generation Method

This alert was generated by Snort version 2.2.0 with rules dated 2004/08/10.

The following rule generated this alert:

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 NOOP";
content:"aaaaaaaaaaaaaaaaaaaa"; classtype:shellcode-detect; sid:1394; rev:5;)
```

This rule will fire an alert if a packet has data that contains a string of 21 ASCII "a" characters. This is the 5th revision of this rule, and the rule id number is 1394.

There were two source IP addresses that sent packets which caused these alerts. 24.86.79.191 sent one packet and 24.95.218.7 sent 12. I am going to show my analysis for the packets from 24.95.218.7, but I also examined the packet from 24.86.79.191 and had the same result.

Here is one of the alerts generated by this rule:

```
[**] [1:1394:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/30-22:38:57.626507 24.95.218.7:7875 -> 207.166.87.157:62871
TCP TTL:113 TOS:0x0 ID:31048 IpLen:20 DgmLen:114 DF
***AP*** Seq: 0xA94CEE1F  Ack: 0x6D157309  Win: 0x40B4  TcpLen: 20
```

I examined the payload of the packets which caused the alerts. I found that they contained the string "Scary Movie - Wazzaaaaaaaaaaaaaaaaaaaaah".

```
000 : 00 00 00 00 00 00 53 0A 00 00 00 F1 39 05 00 D4   ......S.....9...
010 : 80 01 06 33 00 00 00 00 00 53 63 61 72 79 20 4D   ...3.....Scary M
020 : 6F 76 69 65 20 2D 20 57 61 7A 7A 61 61 61 61 61   ovie - Wazzaaaaa
030 : 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61   aaaaaaaaaaaaaaaa
040 : 61 68 00 75 72 6E 3A 00 00 00                     ah.urn:...
```

I looked for other traffic related to the source IP address 24.95.218.7 and found that Snort had logged one packet outbound from the University network less than a second before the incoming packets had been logged. The ports and IP addresses used for this packet are the same as the ones used for the incoming packets, so they are most likely part of the same TCP connection. Here is the alert for the outgoing packet:

```
[**] [1:556:5] P2P Outbound GNUTella client request [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
10/30-22:38:00.076507 207.166.87.157:62871 -> 24.95.218.7:7875
TCP TTL:124 TOS:0x0 ID:45753 IpLen:20 DgmLen:372 DF
***AP*** Seq: 0x6D119EE3  Ack: 0xA94B16F7  Win: 0x4470  TcpLen: 20
```

This outgoing packet was alerted on by the rule:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"P2P Outbound GNUTella client request";
flow:to_server,established; content:"GNUTELLA CONNECT"; depth:40; classtype:policy-
violation; sid:556; rev:5;)
```

This rule looks for any packets from $HOME_NET going to $EXTERNAL_NET on any port. The default Snort configuration has $HOME_NET and $EXTERNAL_NET both set to "any", so this rule will create an alert for any packet with the string "GNUTELLA CONNECT" in its payload. The depth:40; option means that this string must be located within the first 40 bytes of the payload. This is the 5th revision of this rule, and the rule ID number is 556.

Here is the payload for this packet:

```
000 : 47 4E 55 54 45 4C 4C 41 20 43 4F 4E 4E 45 43 54   GNUTELLA CONNECT
010 : 2F 30 2E 36 0D 0A 55 73 65 72 2D 41 67 65 6E 74   /0.6..User-Agent
020 : 3A 20 4D 6F 72 70 68 65 75 73 20 32 2E 30 2E 31   : Morpheus 2.0.1
030 : 2E 38 0D 0A 58 2D 55 6C 74 72 61 70 65 65 72 3A   .8..X-Ultrapeer:
040 : 20 46 61 6C 73 65 0D 0A 50 4F 4E 47 2D 43 41 43    False..PONG-CAC
050 : 48 49 4E 47 3A 20 30 2E 31 0D 0A 58 2D 4D 59 2D   HING: 0.1..X-MY-
060 : 41 44 44 52 45 53 53 3A 20 58 58 58 58 58 58 58   ADDRESS: XXXXXXX
070 : 2E 35 30 2E 31 32 30 3A 37 39 38 38 0D 0A 58 2D   .50.120:7988..X-
080 : 54 72 79 3A 20 31 32 2E 32 32 39 2E 32 34 37 2E   Try: 12.229.247.
090 : 31 37 33 3A 36 37 36 34 2C 31 36 39 2E 32 35 34   173:6764,169.254
0a0 : 2E 31 35 33 2E 32 30 38 3A 36 33 35 33 2C 31 36   .153.208:6353,16
0b0 : 39 2E 32 35 34 2E 39 38 2E 31 39 35 3A 39 39 30   9.254.98.195:990
0c0 : 31 2C 31 34 39 2E 31 36 39 2E 35 38 2E 31 34 39   1,149.169.58.149
0d0 : 3A 35 34 38 31 2C 31 33 34 2E 31 33 39 2E 32 31   :5481,134.139.21
0e0 : 34 2E 32 31 3A 36 30 31 32 2C 36 35 2E 39 36 2E   4.21:6012,65.96.
0f0 : 36 31 2E 31 30 31 3A 35 33 37 38 2C 31 36 39 2E   61.101:5378,169.
100 : 32 35 34 2E 39 38 2E 31 39 35 3A 39 39 30 31 2C   254.98.195:9901,
110 : 36 38 2E 39 37 2E 31 35 38 2E 31 36 37 3A 36 33   68.97.158.167:63
120 : 34 36 2C 36 36 2E 31 35 37 2E 39 38 2E 33 3A 36   46,66.157.98.3:6
130 : 33 34 36 2C 32 31 36 2E 32 34 39 2E 31 37 32 2E   346,216.249.172.
140 : 32 30 32 3A 37 35 38 31 0D 0A 0D 0A               202:7581....
```

The internal host is connecting to the Gnutella (http://www.gnutella.com/) network using the Morpheus (http://www.morpheus.com/) client. I believe that the string "Scary Movie – Wazzaaaaaaaaaaaaaaaaaaaaah" from the incoming packets refers to a file name which was available for download on the Gnutella network. I researched this on Google (http://www.google.com) and found a video clip from the year 2000 film "Scary Movie" (http://imdb.com/title/tt0175142/) that seems like it could be the file which was available on Gnutella. Here is a link with more details: http://www.worldclique.ch/glinz/downloads.htm.

The x86 NOOP alerts were false positives. There was not a buffer overflow attempt. See the correlations section for some other false positives relating to x86 NOOP alerts.

## Address Spoofing Probability

The addresses were not spoofed. There is already an established TCP connection between the two hosts involved. If the external host had been spoofing its IP Address, a TCP three-way handshake would have failed. Without a TCP connection, the Gnutella client applications on each host would not exchange the data which was captured in these alerts.

## Attack Mechanism

Gnutella is a peer-to-peer file sharing network. There are many client applications available for connecting to this network. A list of some clients can be found at http://www.infoanarchy.org/wiki/index.php/Gnutella_Clients. http://computer.howstuffworks.com/file-sharing3.htm has a good explanation of how a Gnutella search works:

*How a Gnutella client finds a song*
*Given that there is no central server to store the names and locations of all the available files, how does the Gnutella software on your machine find a song on someone else's machine? The process goes something like this:*

** You type in the name of the song or file you want to find.*
** Your machine knows of at least one other Gnutella machine somewhere on the network. It knows this because you've told it the location of the machine by typing in the IP address, or because the software has an IP address for a Gnutella host pre-programmed in. Your machine sends the song name you typed in to the Gnutella machine(s) it knows about.*
** These machines search to see if the requested file is on the local hard disk. If so, they send back the file name (and machine IP address) to the requester.*
** At the same time, all of these machines send out the same request to the machines they are connected to, and the process repeats.*
** A request has a TTL (time to live) limit placed on it. A request might go out six or seven levels deep before it stops propagating. If each machine on the Gnutella network knows of just four others, that means that your request might reach 8,000 or so other machines on the Gnutella network if it propagates seven levels deep.*

After the results are returned to the searcher, the user can instruct the client software to download the files.

Information on the Morpheus client used on internal host can be found at http://www.morpheus.com/faq.html.

## Correlations

I have not seen this detect analyzed before, but there is documentation of similar false positives with NOOP rules. Robert David Graham has documented some ways that a false positive can occur: http://archives.neohapsis.com/archives/sf/ids/2002-q2/0029.html.

SANS student Terry MacDonald saw a false positive with a previous revision of the same rule that caused the alert I examined. Information is in his GCIA practical http://www.giac.org/practical/GCIA/Terry_MacDonald_GCIA.pdf.

Dave Love saw some false positives with a different NOOP rule. His report can be viewed at http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00346.html.

Link to Snort Signature Database for this alert: http://www.snort.org/snort-db/sid.html?sid=1394.

Evidence of Active Targeting

This traffic was definitely actively targeting the internal host. It was part of an established connection between two peers on the Gnutella network. The external host is using the existing connection to send data directly to the internal one.

Severity

**Criticality-3** Since this alert was caused by some Gnutella traffic, the target is probably a user's workstation on the network. While a workstation is not an extremely critical system, having a compromised system on your network is still a serious problem.

**Lethality-1** This alert was determined to be a false positive, so there is not a real attack happening here.

**System Countermeasures-1** A Gnutella client (Morpheus.2.0.1.8) is installed on the system, and there is Gnutella traffic both to and from the internal host. There is nothing which suggests that the system is trying to prevent this traffic.

**Network Countermeasures-2** There is Gnutella traffic leaving and entering the network, so there does not appear to be anything on the network limiting this traffic. I gave this a 2 rather than a 1 because the IDS generated an alert.

Severity will be calculated using the formula:

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

severity = (3 + 1) - (1 + 2)

**severity = 1**

## 3.2 Detect Two - BACKDOOR Q access

## Description

This attack sends packets to a host which has already been compromised with the Q backdoor, giving Q instructions on what to do.

Les Gordon's article "On Q" (http://www.whitehats.ca/main/publications/external_pubs/Q-analysis/Q-analysis.html) provides the following explanation of the Q backdoor.

*Q is a primarily Unix-based remote-access tool that provides stealth capabilities to make it's presence less obvious both on the host, and in network traffic. The product could conceivably be used by sysadmins, but I think it's far more likely to appeal to those with less pure motives. The latest version at time of writing (2.4) may be able to be compiled and run on W32 systems as well, although I didn't try this myself. All versions of Q are compiled as a client/server pair. Different client/server pairs cannot establish usable sessions with each other by default. "qd" is the server portion of Q. "qs" is used to send covert messages to the qd server. These messages are used to remotely execute privileged commands, invoke single-use encrypted remote shell processes, and to set up port redirection or "bounce" servers on the target host.*

CVE candidate CAN-1999-0660 deals with backdoors in general http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0660.

## Reason for Selection

I chose this as one of the three most critical detects because if it was successful, there could be multiple compromised hosts on the network. In that case, I would like to find out everything that I could about the attack in order to deal with those hosts. I also felt that this traffic should not have made it on to the network in the first place. I wanted to learn more about it so that I could recommend to the University how and why it should be blocked.

## Detect Generation Method

This alert was generated by Snort version 2.2.0 with rules dated 2004/08/10.

The following rule generated this alert:

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; dsize:>1;
flags:A+; flow:stateless; reference:arachnids,20
3; classtype:misc-activity; sid:184; rev:6;)
```

In order for an alert to be generated by this rule, there must be a TCP packet with a source IP address in the 255.255.255.0/24 network. The destination IP address must be in a network listed in the $HOME_NET variable, which is set to "any" by default. This packet must at least have the ACK flag set and must contain greater than 1 byte of data in the payload. The flow:stateless option tells Snort that the TCP state of the packet does not matter. This is revision number 6 of this rule, and the rule ID is 184.

Here is one of the alerts generated by this rule:

```
[**] [1:184:6] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
10/30-22:54:07.176507 255.255.255.255:31337 -> 207.166.183.55:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS203]
```

45 addresses on the University network were targeted by this attack. Each address was sent exactly one packet, except for 207.166.124.130 which received two. All of the packets had a source IP address of 255.255.255.255. The source port was 31337, and the destination port was 515. Each packet had a 3 byte payload of the string "cko". Both the ACK and RST flags were set on every packet.

Here is a sample packet.

```
16:07:04.816507 IP (tos 0x0, ttl  15, id 0, offset 0, flags [none], proto 6, length: 43,
bad cksum adb9 (->636f)!) 255.255.255.255.31337 > 207.166.120.184.515: R [bad tcp cksum
62e1 (->1897)!] 0:3(3) ack 0 win 0 [RST cko]
        0x0000:  4500 002b 0000 0000 0f06 adb9 ffff ffff  E..+............
        0x0010:  cfa6 78b8 7a69 0203 0000 0000 0000 0000  ..x.zi..........
        0x0020:  5014 0000 62e1 0000 636b 6f00 0000       P...b...cko...
```

## Address Spoofing Probability

I believe that these packets were spoofed. The IP address 255.255.255.255 is part of the 255.0.0.0/8 network, which is listed as "Reserved" by IANA (http://www.iana.org/assignments/ipv4-address-space), so nobody should be using it on their network as a source address. Furthermore, RFC3330 (ftp://ftp.rfc-editor.org/in-notes/rfc3330.txt) states the following:

> *240.0.0.0/4 - This block, formerly known as the Class E address space, is reserved. The "limited broadcast" destination address 255.255.255.255 should never be forwarded outside the (sub-)net of the source. The remainder of this space is reserved for future use.*

There are other suspicious things about these packets. This gives additional support to the possibility of an altered source IP. First there is the source port of 31337, which is slang used by some hackers for the word "elite", and is a port

used by multiple trojans, the ACK+RST flags is set, possibly to bypass routers and firewalls which are set to allow "established" connections, and it is unusual that the IP identification number is set to 0 on every packet.

## Attack Mechanism

Since the attacker is using a source address that is spoofed, the packets must be able to serve a purpose without having to send data back to the attacker. The Les Gordon article mentioned earlier examined the Q backdoor in great detail and tells much about the attack mechanism used here.

Q is designed to listen to all packets destined for the infected host. The attacker crafts a packet with specific parameters and sends it to the host. The attack can be sent via UDP, TCP, or ICMP. Once the Q server receives the control packet from the attacker, it will perform a specific function. The packets seen here do not appear to contain the options listed in the article, so this may be a modified version or another backdoor that is Q-like.

Al Maslowski-Yerges saw similar traffic in two weeks of analyzed logs while working on his GCIA practical http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf.

There is also some thought that this could have something to do with IRC, as discussed at the following URL.
http://archives.neohapsis.com/archives/incidents/2001-05/0039.html

## Correlations

Les Gordon's article "On Q":
http://www.whitehats.ca/main/publications/external_pubs/Q-analysis/Q-analysis.html

CAN-1999-0660:http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0660

Discussion of possible IRC link:
http://archives.neohapsis.com/archives/incidents/2001-05/0039.html

Al Maslowski-Yerges GCIA Practical:
http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf

Link to Snort Signature Database for this rule:
http://www.snort.org/snort-db/sid.html?sid=184.

## Evidence of Active Targeting

The attacker is sending these packets to seemingly random hosts on the University network, so I do not think a specific host is being actively targeted.

The attacker does appear to know what he is looking for, and these packets have been crafted to perform some specific function.

The attacker may have some general information regarding a Q server installed somewhere on the University network, and is using the Q client in an attempt to find it.

Severity

---

Severity will be calculated using the formula:

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

**Criticality-4** This attack targets multiple addresses on the University network. It is unknown what these targeted addresses are used for. These addresses could not even be in use, so I do not want to give this the highest criticality rating. I rated this attack a 4 to keep the severity high so that it is not ignored. If more information becomes available this rating could be changed.

**Lethality-5** A backdoor usually means that the attacker gains root access to the attacked system. For that reason I chose a very high lethality rating.

**System Countermeasures-1** Nothing is known about the target systems, so I am unable to determine that they have anything in place that would stop this attack. If other information becomes available, such as the existence of host-based firewalls, this rating could be raised.

**Network Countermeasures-2** This fact that this traffic was allowed in to the network causes me to question the countermeasures that are in place. This traffic is easily blocked on any Cisco router or firewall, and should be done so immediately. I gave this a 2 rather than a 1 because the IDS generated an alert.

severity = (4 + 5) - (1 + 2)

**severity = 6**

---

## 3.3 Detect Three - SCAN SOCKS Proxy attempt

### Description

In this attack, the attacker is looking for proxy servers on the University network. A proxy server acts as a data relay. It takes in packets from a client, and then sends those packets to the destination address specified by the client, making them appear to have originated from the proxy server's IP address. If the proxy server is behind a firewall, an attacker could use a proxy server to gain further access into a protected network. It could also be used to anonymize attacks on other external IP addresses.

### Reason for Selection

I chose to examine this attack because it accounted for the highest number of alerts for incoming traffic. If this type of attack could be understood and eliminated, it would lessen the work load on analysts and allow them to more easily find and investigate other, more serious attacks.

### Detect Generation Method

This alert was generated by Snort version 2.2.0 with rules dated 2004/08/10.

At first this attack did not show up in Snort. After examining the traffic with tcpdump, I saw that port 1080 was the top targeted port on the internal network. Since the log file only contains packets that generated alerts on the original Snort sensor, there had to be a rule that would detect this attack. I looked for any rules having to do with port 1080 and found the following in the deleted.rules file:

```
#-------------
# DELETED RULES
#-------------
# These signatures have been deleted for various reasons, but we are keeping
# them here for historical purposes.
…
…
alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS Proxy attempt"; flags:S,12;
flow:stateless; reference:url,help.undernet.org/proxyscan/; classtype:attempted-recon;
sid:615; rev:9;)
```

This rule will generate an alert when a TCP packet is sent from any port on an IP addresses specified in the $EXTERNAL_NET variable to port 1080 on any IP addresses specified in the $HOME_NET variable. Both variables include all IP addresses in the default Snort configuration. The packet must have the SYN

flag set, and the values of the reserved bits are ignored. There is a reference to the URL help.undernet.org/proxyscan, which should have more information about this kind of attack. The flow:stateless option means that the stream processor The Snort rule ID number is 615, and this is version 9 of the rule.

The reason given for removing this rule was “because this rule sucks”. It was probably removed for causing a lot of false positives due to the fact that it is only looking at the destination port and has no content or other advanced checking.

Here is an alert generated by this rule:

```
[**] [1:615:9] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/30-18:04:20.246507 216.77.216.150:38670 -> 207.166.185.2:1080
TCP TTL:49 TOS:0x0 ID:16218 IpLen:20 DgmLen:40
******S* Seq: 0x63D74BFD  Ack: 0x63D74BFD  Win: 0x400  TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]
```

This is one of the packets that caused an alert. As you can see, it is a SYN packet sent to port 1080, which is what the rule is looking for.

```
2002-10-31 03:01:06.016507 IP 216.77.216.150.9546 > 207.166.145.215.1080: S
784161074:784161074(0) win 1024
```

## Address Spoofing Probability

I do not think the addresses in this attack were spoofed. If an attacker wanted to find and use an open proxy, they would need to establish a TCP connection to the proxy server. That would be very difficult if a spoofed source IP was used.

## Attack Mechanism

I am going to examine these attacks coming from three different source IP addresses. These may be three different attackers, but it is probable that the attacker has changed his IP address during the attack.

The first IP address is 216.77.216.150. This address sent 133 packets between 00:00:01 and 09:29:06. There was about a 4 minute interval between each packet. After this first IP stopped sending packets, there was a gap of about 20 minutes until 09:50:36, and a new IP address, 216.77.216.176, started sending them. This continued at the same 4 minute interval until 12:21:29. This address sent 36 packets total. There was another gap, this time of about 1 hour until 13:24:30, when 216.77.209.189 started sending the packets. This address sent 250 packets, the last of which was seen at 23:58:39 just before logging for this day ended. A total of 419 internal addresses were targeted with a single packet, and there were no overlaps between the three source IP addresses.

When investigating the three addresses, I found that they all appear to be used by bellsouth.net DSL users. Many DSL providers assign a dynamic IP address to the user at the time of connection. The attacker could be disconnecting and reconnecting his DSL connection before continuing the scan. The packets were all very similar, with matching TTL values and window sizes.

Reverse DNS lookups from http://www.dnsstuff.com:

```
216.77.216.150 PTR record: adsl-77-216-150.aec.bellsouth.net.
216.77.216.176 PTR record: adsl-77-216-176.aec.bellsouth.net.
216.77.209.189 PTR record: adsl-77-209-189.aec.bellsouth.net.
```

Whois lookup from http://arin.net:

```
OrgName:    BellSouth.net Inc.
OrgID:      BELL
Address:    575 Morosgo Drive
City:       Atlanta
StateProv:  GA
PostalCode: 30324
Country:    US

ReferralServer: rwhois://rwhois.eng.bellsouth.net:4321

NetRange:   216.76.0.0 - 216.79.255.255
CIDR:       216.76.0.0/14
NetName:    BELLSNET-BLK5
NetHandle:  NET-216-76-0-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
```

This attacker appears to be trying to evade detection, both by using a slow scanning rate, not scanning target addresses sequentially, and by periodically changing source IP addresses. He is probably using a script to do the scanning, which randomizes the scanned IP range and sets the 4 minute interval.

SANS student Steve Breault documented a similar attack in his GCIA practical located at http://www.giac.org/practical/GCIA/Steve_Breault_GCIA.pdf.

## Correlations

Steve Breault documented a different SOCKS proxy scan:
http://www.giac.org/practical/GCIA/Steve_Breault_GCIA.pdf.

Here is a link to the Snort Signature Database for this rule:
Snort rule: http://www.snort.org/snort-db/sid.html?sid=615.

## Evidence of Active Targeting

The attacker is not targeting a specific host. This is a scan of what looks like random addresses on the University network. The attacker is sending one SYN packet to each address in an attempt to locate a host responding on port 1080. They are probably looking for an open proxy server.

Severity

Severity will be calculated using the formula:

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

**Criticality-4** A proxy server can be an aggregation point for all data leaving a network, so it can be critical to network operations.

**Lethality-2** This attack is just a scan. The only danger would be that the attacker learns some information about systems on the network. With that information, the attacker could perform more serious attacks, but those would be rated individually and do not apply to this attack.

**System Countermeasures-1** There is nothing known about the attacked IP addresses, so until further information can be provided I am giving this the lowest rating.

**Network Countermeasures-2** The only known countermeasure is the IDS which logged the attack. Therefore I am giving this a 2 rating. I would need more information about router ACLs or firewall in order to consider raising this.

severity = (4 + 2) - (1 + 2)

**severity = 3**

## *4. Network Statistics*

### 4.1 Top Talkers

I chose for top talkers the IP addresses that sent the highest numbers of packets.

Top external talkers:

| # of packets sent | IP Address |
|---|---|
| 259 | 216.77.209.189 |
| 133 | 216.77.216.150 |
| 46 | 255.255.255.255 |
| 43 | 68.36.170.9 |
| 37 | 128.167.120.13 |

The top two external talkers, 216.77.209.189 and 216.77.209.189, were involved in the SOCKS proxy scan I examined earlier. 255.255.255.255 was the address involved in the Q backdoor attack, which I also examined earlier. 68.36.170.9 attempted some web attacks against the internal web server 207.166.87.40, and 128.167.120.13 appears to be a web server replying to an internal NAT host with an invalid HTTP version string.

Top internal talkers:

| # of packets sent | IP Address |
|---|---|
| 3709 | 207.166.87.157 |
| 5 | 207.166.87.40 |

There were only two internal source addresses logged by Snort.
207.166.87.157 is a NAT address used by many different hosts.
207.166.87.140 is web server which sent out some HTTP error messages.

## 4.2 Top Targeted Ports

The top targeted ports are the ones which were sent the highest numbers of packets.

The top five ports targeted by external sources were:

| # of packets | Port |
|---|---|
| 431 | 1080 |
| 161 | 9511 |
| 86 | 80 |
| 46 | 515 |
| 18 | 53 |

Port 1080 was targeted in the SOCKS proxy scan I examined earlier. Port 9511 was used for Gnutella traffic. Port 80 was targeted by various web attacks. Port 515 was the target used in the Q backdoor attack I examined earlier, and port 53 may have been used for DNS zone transfers.

The top five ports targeted by internal sources were:

| # of packets | Port |
|---|---|
| 2191 | 80 |
| 67 | 6347 |
| 38 | 1863 |
| 30 | 6348 |
| 19 | 6349 |

Port 80 had many alerts caused by cookies used by spyware. Ports 6347, 6348, and 6349 were used for Gnutella connections, and port 1863 was used for MSN Messenger chat.

## 4.3 Three Most Suspicious External Source Addresses

**1.** 255.255.255.255 – I chose this as one of the most suspicious addresses because it was the source address used for the Q backdoor attack I examined earlier. The address is obviously spoofed, and the attacker is trying to locate a backdoor on the University network. This address is part of a reserved network, and should not be used as a source address.

Registration information from http://www.arin.net/:

*OrgName: Internet Assigned Numbers Authority*
*OrgID: IANA*
*Address: 4676 Admiralty Way, Suite 330*
*City: Marina del Rey*
*StateProv: CA*
*PostalCode: 90292-6695*
*Country: US*

*NetRange: 240.0.0.0 - 255.255.255.255*
*CIDR: 240.0.0.0/4*
*NetName: RESERVED-240*
*NetHandle: NET-240-0-0-0-0*
*Parent:*
*NetType: IANA Special Use*
*Comment: Please see RFC 3330 for additional information.*
*RegDate:*
*Updated: 2002-10-14*

*OrgAbuseHandle: IANA-IP-ARIN*
*OrgAbuseName: Internet Corporation for Assigned Names and Number*
*OrgAbusePhone: +1-310-301-5820*
*OrgAbuseEmail: abuse@iana.org*

*OrgTechHandle: IANA-IP-ARIN*
*OrgTechName: Internet Corporation for Assigned Names and Number*
*OrgTechPhone: +1-310-301-5820*
*OrgTechEmail: abuse@iana.org*

From what I have read, the Q backdoor is UNIX-based, so the attacker's operating system is probably UNIX or linux.

**2.** 68.36.170.9 – This attacker is directly targeting a known web server on the University network with various web-related attacks. The attacks are coming from what appears to be a cable modem user. The TTL on these packets is 114, which could mean that the attacker is on a windows host.

Whois lookup from http://arin.net:

```
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. NJ-NORTH-1 (NET-68-36-0-0-1)
                                  68.36.0.0 - 68.36.255.255
```

Reverse DNS lookup from http://www.dnsstuff.com:

```
68.36.170.9 PTR record: pcp02172463pcs.verona01.nj.comcast.net.
```

**3.** 216.77.216.150, 216.77.216.176, and 216.77.209.189 – These are the addresses which I examined in my earlier analysis of SOCKS proxy scanning. I have included the registration information in that analysis. I attempted to identify the host operating system used, but results from both p0f and my own analysis were inconclusive, but the TTL of 49 could mean that UNIX or Linux is in use.

## *5. Correlations - GCIA, CERT, BugTraq, Etc.*

Robert David Graham NOOP false positives:
http://archives.neohapsis.com/archives/sf/ids/2002-q2/0029.html.

Terry MacDonald NOOP detect:
http://www.giac.org/practical/GCIA/Terry_MacDonald_GCIA.pdf.

Dave Love NOOP false positives: http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00346.html.

Link to the Snort Signature Database for SHELLCODE x86 NOOP:
http://www.snort.org/snort-db/sid.html?sid=1394.

Les Gordon's article "On Q":
http://www.whitehats.ca/main/publications/external_pubs/Q-analysis/Q-

analysis.html

CAN-1999-0660:http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0660

Discussion of possible IRC link:
http://archives.neohapsis.com/archives/incidents/2001-05/0039.html

Al Maslowski-Yerges GCIA Practical:
http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf

Link to the Snort Signature Database for Q backdoor:
http://www.snort.org/snort-db/sid.html?sid=184.

Steve Breault documented a different SOCKS proxy scan:
http://www.giac.org/practical/GCIA/Steve_Breault_GCIA.pdf.

Link to the Snort Signature Database for SOCKS proxy attempt:
Snort rule: http://www.snort.org/snort-db/sid.html?sid=615.

## *6. Potentially Compromised Internal Hosts*

- There are multiple hosts on the network that are sending out spyware related traffic. They are all behind the NAT address 207.166.87.157, so I am unable to identify them directly.
- The hosts involved in the Q backdoor scan discusses earlier may be compromised. I only had the attack packets, so I was unable to determine if any of them responded to this attack.
- There are hosts on the network using Gnutella. Many viruses and Trojans are spread through peer-to-peer file sharing networks. There are also copyright infringement matters to be concerned with. The Gnutella traffic came from behind the NAT address 207.166.87.157, so again I am unable to identify a specific host.
- The web server 207.166.87.40 was the target of some web attacks. This server should be checked for compromise.

## *7. Defensive Recommendations*

I have some recommendations that will improve the security of this network and also make analyzing future IDS alerts easier and less time consuming.

- Perform proper filtering at the network border. Many alerts have been caused by traffic that does not need to be allowed to enter the network. I recommend

that all routers and firewalls be examined and stronger filtering be put in place. Only necessary traffic should be allowed in, and it may be wise to stop or limit peer-to-peer file sharing. For ingress filtering, I recommend adding to your border routers the access list located at http://www.cymru.com/Documents/bogon-dd.html#dd-acl-agg. This will block all networks which have been given reserved status by IANA.
- Tune the existing IDS to eliminate unwanted alerts and false positives. One example is the x86 NOOP alerts I analyzed earlier. The $SHELLCODE_PORTS variable could be modified to ignore ports used for file transfers.
- Place additional IDS on the internal NAT network in order to identify problem hosts. This new IDS server would be tuned for operation on the NAT network.
- Inform users about spyware privacy issues and help them to eliminate spyware on their systems. There are programs such as Ad-Aware (http://www.lavasoft.de) which can find and remove spyware from a host.
- Set up a system to capture all packets entering and leaving the network. This would help immensely when analyzing IDS events. The IDS will only capture packets that cause alerts, so some other method of capture is required. I recommend using IDABench http://idabench.ists.dartmouth.edu/.
- Timely updating and patching of systems needs to be a priority. Many attacks are successful only because the target is running old software versions. The web server 207.166.87.40 should be examined to see if it is still running the same OS and version of apache that it was two years ago. If so, the latest patches and upgrades should be installed.
- If not currently being done, I recommend that a vulnerability scanner be used. Nessus (http://www.nessus.org/) is open-source and can be obtained at no cost to the University. This will allow possible threats to be eliminated before they become an issue.
- I recommend the use of host-based firewalls on all servers and workstations. This is an extra layer of defense that can prevent attacks that find a way to bypass your network firewalls. For Linux-based hosts, I recommend the use of iptables (http://www.netfilter.org/). I have had success in the past using ZoneAlarm (http://www.zonelabs.com/) on Windows hosts.

## Part III: Analysis Process

My analysis was performed on a HP system with an Intel Celeron 533MHz processor and 256MB of RAM. The operating system used was Fedora Core 2 with kernel version 2.6.5. Tcpdump version 3.8.2 was used for analysis. To generate the alerts I used Snort version 2.2.0 with the included rules which were dated 2004/08/10. Snort was built with the --with-mysql option to allow logging to a MySQL database. MySQL version 3.23.58 was used. For analyzing the Snort alerts I used BASE (http://base.secureideas.net/) version 1.0. BASE is a web-based analysis console based off of ACID (http://acidlab.sourceforge.net/). I used Apache HTTP Server version 2.0.51 to run BASE. I connected to BASE using a Dell laptop with an Intel Pentium 4 1.6GHz processor and 384MB of

RAM running Microsoft Windows XP Professional. The web browser I used was Mozilla Firefox version 1.0. Microsoft Visio 2003 was used to produce the network topology map and the link graph. I used p0f (http://lcamtuf.coredump.cx/p0f.shtml) version 2.0.5 to attempt some operating system identification.

I installed all of the applications, then created the snort database and database user. I used the command `mysql -D snort -u root -p < create_mysql` to create the database tables using the script that came with Snort. I edited the BASE configuration file `base_conf.php`, and then connected to BASE using my web browser to make sure it could connect to the database.

Next I made some modifications the `snort.conf` file. I configured logging to the MySQL database, enabled all of the snort rulesets that come disabled by default, configured http_inspect, and disabled the stream4 preprocessors. My reason for enabling the rulesets and using http_inspect was to try and maximize the number of alerts generated. I disabled the stream4 preprocessors because the log file only contains packets which caused alerts. The packets showing the TCP three-way handshakes were not included, so the stream reassembly would not work properly, and possibly cause Snort to miss some alerts.

http_inspect configuration:

```
preprocessor http_inspect: global \
    iis_unicode_map /etc/snort/conf/unicode.map 1252

preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 1080 8180 3128 } oversize_dir_length 500
```

MySQL database logging:

```
output database: log, mysql, user=XXXXXX password=XXXXXX dbname=snort host=localhost
```

I ran snort with the command:

```
snort -dr 2002.9.31 -k none -c /etc/snort/conf/snort.conf -l log
```

The options I used are as follows:
**-d** was used to dump the application layer data
**-r 2002.9.31** was used to read packets from the file 2002.9.31
**-k none** was used to disable checksum verification
**-c /etc/snort/conf/snort.conf** tells Snort to use my configuration file
**-l log** tells Snort the directory for plain text logging

After Snort ran, I had the alert data in both plain text and MySQL database forms. BASE made it very easy to view and find the data I needed to do my analysis. I used http://www.google.com to do all of my research and used tcpdump along with the UNIX programs awk, cut, less, grep, and sort to do some additional analysis.

UNIX programs used:

Tcpdump - http://www.tcpdump.org/tcpdump_man.html
Awk - http://unixhelp.ed.ac.uk/CGI/man-cgi?awk
Cut - http://unixhelp.ed.ac.uk/CGI/man-cgi?cut
Less - http://unixhelp.ed.ac.uk/CGI/man-cgi?less
Grep - http://unixhelp.ed.ac.uk/CGI/man-cgi?grep
Sort - http://unixhelp.ed.ac.uk/CGI/man-cgi?sort

## References


IEEE MAC OUI listings: http://standards.ieee.org/regauth/oui/oui.txt

Honeynett Progect fingerprinting traces:
http://project.honeynet.org/papers/finger/traces.txt

Robert David Graham on x86 NOOP alerts:
http://archives.neohapsis.com/archives/sf/ids/2002-q2/0029.html

Smashing The Stack For Fun And Profit by Aleph One:
http://www.phrack.org/show.php?p=49&a=14

Scary Movie: http://imdb.com/title/tt0175142/

Scary movie clip: http://www.worldclique.ch/glinz/downloads.htm

Gnutella Client List: http://www.infoanarchy.org/wiki/index.php/Gnutella_Clients

How Gnutella works: http://computer.howstuffworks.com/file-sharing3.htm

Terry MacDonald GCIA Practical
http://www.giac.org/practical/GCIA/Terry_MacDonald_GCIA.pdf

Dave Love SHELLCODE x86 NOOP analysis: http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00346.html

SID 1394 - SHELLCODE x86 NOOP. Snort Signature Database.
http://www.snort.org/snort-db/sid.html?sid=1394

Les Gordon "On Q" article
http://www.whitehats.ca/main/publications/external_pubs/Q-analysis/Q-analysis.html

CVE candidate CAN-1999-0660 http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0660

IANA reserved networks http://www.iana.org/assignments/ipv4-address-space

RFC3330 ftp://ftp.rfc-editor.org/in-notes/rfc3330.txt

Al Maslowski-Yerges GCIA Practical:
http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf

Discussion of possible IRC cause of Q backdoor alerts:
http://archives.neohapsis.com/archives/incidents/2001-05/0039.html

SID 184 - BACKDOOR Q access." Snort Signature Database.
http://www.snort.org/snort-db/sid.html?sid=184

DNSStuff DNS tools: http://www.dnsstuff.com

ARIN Whois lookup http://ws.arin.net

Steve Breault GCIA Practical:
http://www.giac.org/practical/GCIA/Steve_Breault_GCIA.pdf

SID 615 - SCAN SOCKS Proxy attempt. Snort Signature Database.
http://www.snort.org/snort-db/sid.html?sid=615

List of reserved networks in Cisco ACL format:
http://www.cymru.com/Documents/bogon-dd.html#dd-acl-agg