Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 3.4

Amanda Meyer

Submitted: April 17, 2004

Summary:

The following paper contains my GIAC Certified Intrusion Analyst (GCIA) practical assignment submission. Included within this document are a white paper on the subject of securing internal servers using a host-based implementation of Snort, three practical detect submissions, as well as a security log analysis paper (entitled "Analyze This") which provides an in-depth analysis of the University of Maryland Baltimore County's IDS files. References for each section can be found at the end of the respective section.

The Soft, Squishy Center:
Using Snort to Detect Internal Server Attacks in a Windows Environment

Introduction

It has been stated by many different security authorities that internal employees make up at least half (in most cases, more) of the security threats facing the average organization today. [1,2,3] It is not a difficult theory to argue; after all, internal users are more familiar with the network and company politics than any outside attacker ever could be (we hope). Every morning when Joe User logs into his personal workstation he is presented with a veritable smorgasbord of information your average seciruty administrator would shudder to have exposed to a potential hacker: the domain name, his own username (complete with a valid password to login), the names of all the file and application servers (usually via mapped drives), as well as nearly each and every program used by that organization on a daily basis. His Windows workstation even comes equipped with some very effective reconnaissance tools built-in: ping, tracert, and nslookup are three ready examples.

There are obviously many levels to securing a Windows networking environment, and it is out of the scope of this paper to discuss all of them. What this paper will focus on is defending against the inevitable weaknesses involved with allowing anyone, especially your own users, access to the mission critical servers on the network…which is, of course, the whole point of networking.

More specifically, this paper will focus on designing a Host-Based implementation of Snort that will help detect unauthorized network access attempts to Windows 2000 Server domain controllers and member servers. Since performance is a definite consideration for any server that actually provides services, this paper will also focus on tuning the rule set used by Snort to be tighter and more lightweight than your default Network IDS rule set would be. As well, some suggestions for deploying Snort to mission critical servers in your small-medium sized business will be included, in addition to methods for log collection and alerting mechanisms.

Where to Begin

Before beginning any host-based installation of Snort, it should be confirmed that the host in question has been properly hardened and secured. In the Windows 2000 world, this usually involves installing the latest service packs and hotfixes, shutting off the many unnecessary services that are enabled by default, securing all file share permissions, as well as seeing to the physical security of the box itself (in other words, don't leave it sitting in the employee break room). Though it is out of the scope of this paper to go into specific detail of how to harden and secure a Windows box, the SANS institution offers a course specifically geared towards this subject (Securing Windows); as well, there are many white papers available on securing Windows 2000 in the SANS InfoSec Reading Room. [4]

---

[1] Andrews, Peter
[2] TechRepublic
[3] Waveset
[4] The SANS Institute. Windows 2000 Issues.

Why not just a NIDS on the Subnet?

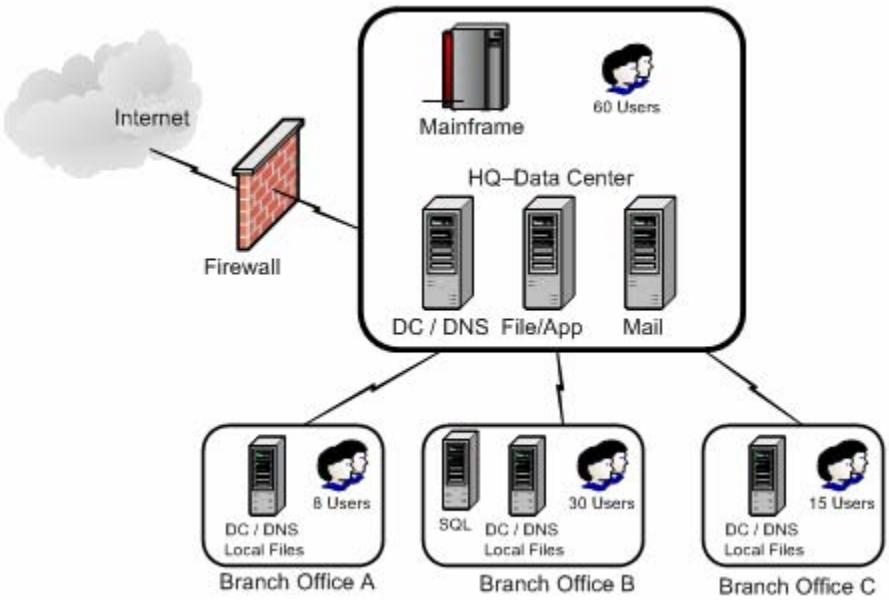
For many enterprise organizations, deploying a Network Intrusion Detection System on each subnet they want to monitor makes a lot of sense: you only have one installation, one set of logs, and one set of rules per subnet to monitor. One sniffer can monitor the entire network and report back what they need to know: who's attacking what. One limitation to this implementation is that the rule set used on an entire subnet generally has to be very broad in order to accommodate all of the services offered on that network, and it is possible to miss lower profile events (such as an attempted telnet to a server that does not support telnet). Although variables can be configured in the rule set to accommodate closer inspection of certain IP addresses, this often leads to more rule maintenance as the network grows and changes, and also requires more processing power, storage space and memory…in other words, more $.

In the average small to medium sized business, there are fewer subnets, fewer servers, and more importantly fewer staff members available to monitor the network (as well as less $). Having less time and resources to dedicate to security and rule maintenance often forces that organization to make certain compromises.

I will use my organization as an example network (see Figure 1 below). We are a medium-sized bank with 12 branch offices located throughout the state. Our Data Center is located the northern part of the state and it is here that we own our largest and most diverse subnet. In the rest of our branch offices there are typically only 10-30 workstations and 1-2 servers providing local domain authentication, name resolution, and file sharing. In an organization of roughly 250 employees, 20 servers and a Core mainframe, and 300 desktop workstations we have a total IT Staff of four strong (that includes help desk and mainframe administration staff). As one can imagine, that doesn't leave a lot of extra man power for monitoring security logs, especially when considering the fact that all of the servers generate their own event logs, as well as the firewall and routers. We need a solution that is lightweight, easy to deploy, and that does not generate so much log data that it would get archived, but never examined (an ill, but common, fate for log files in smaller organizations).

At the Data Center, there are enough mission critical services running (including our only connection to the Internet) to implement a NIDS (Network IDS)…two actually, one just behind the firewall, and one on a separate segment where our servers reside. We made this decision for two reasons: first, one of our highest priorities is to monitor any potentially malicious traffic that is coming in from external networks (a priority of most organizations). Another is that we also need to more closely monitor access to our mission-critical servers, as well as the servers in our branch offices. The NIDS on the server segment, though it is indeed network based and monitors the entire VLAN, is configured with the same, tighter rule set as our branch office servers (though slightly modified to accommodate the mail server). In this way, we avoid having to implement multiple IDS installations when two will suffice, but are still able to keep a closer eye on our servers in the Data Center.

Figure 1—Sample Network



Note: There are actually 12 Branch offices similar to those above; only 3 are noted here for the sake of brevity.

In regard to the branch offices, it is simply not practical to configure a separate sniffing appliance for each subnet, for many reasons. For one thing, even though Snort doesn't require much hardware, it does require some and that is more than we have available for all 12 offices (budgeting is also out of the scope of this paper). Since the only Internet connection is at the HQ LAN, traffic in the branch offices is fairly restricted; mostly local and to and from the Data Center. It is not possible to realistically expect to be able to monitor all of the log traffic for every workstation in each of these offices (mostly for staffing reasons), so a compromise was made: install a host-based version of Snort on the mission-critical branch servers only, and only monitor traffic into and out of that server, which is the most important asset on each LAN. (These branch servers were judged to be the "most important asset" based on the fact that they provide all of the local domain authentication, file sharing, and name-resolution for that LAN, but are "backed-up" by the servers in the Data Center should they fail). This solution always us to very closely monitor the servers and to tune the rule set to detect any suspicious events (even something as benign as an attempted telnet connection).

We are obviously taking on a certain amount of risk in adopting this security model, in that we are not able to monitor all traffic to and from the workstations and could potentially miss an intrusion event. However, after examining the situation, we decided that the greatest security risk in these branch offices is actually the users themselves…those people that have quite a bit of knowledge about their own LAN (and the company as a whole), and who would know specifically what to attack and where. (This also applies to those users who are misguided enough to "Click Here to Win Money from Bill Gates!!!" and inadvertently start the spread of some worm or spyware.) In addition, we have to take into consideration the fact that we have a high employee turnover rate in the branch offices, thus increasing our risk of sabotage from a disgruntled worker.

In terms of support and recovery, it is actually more difficult to recover one lost branch server than all 10 branch workstations because of technologies such as imaging that are available for workstation recovery. As well, all of the user data is stored on the branch server, not on the local machine, thus increasing the recovery time if a server is lost. (Regular backups of these servers do occur nightly, but it is still possible to lose 24 hours worth of data.)

In order to help eliminate some of the risk involved with not monitoring the workstation traffic, we also chose to purchase a commercially available product called Contego, which facilitates the installation of a point of presence on every workstation and provides the capability to monitor the event and virus logs on those workstations, which alerts us to any possible attacks via a centralized logging console.[5] The Contego appliance is also capable of integrating Snort alerts into its own logging console (which will be discussed in more detail later in this paper).

In reaching this solution, we feel that we have viable security model for detecting intrusion events on the entire network, based on a combination of open-source and commercially available product implementations on both the servers and workstations.

What is Considered Mission-Critical?

The definition of what is considered mission-critical, or vital to the company's business model, will vary within each organization. There are also many factors that determine how much risk a company is willing to acquire, from the type of service that company offers to the company culture itself. Though it is beyond the scope of this paper to discuss risk and asset analysis, SANS also offers a course in Security Auditing Essentials, and excellent resources on the subject in the SANS InfoSec Reading Room. [6] For the purposes of this paper, focus on the server(s) that always produce a phone call from the C.E.O., C.O.O., and/or C.I.O. when they go down, and you're probably on the right track.

Why Snort?

Snort Intrusion Detection system was chosen for many reasons, including the following:

1) It is open-source and free of cost.
2) There is a wide user base and therefore a wide range of support for it in the security community.
3) It has a very flexible and easy to learn rule set.
4) It is lightweight and doesn't require a lot of system resources to operate efficiently.
5) It is compatible with the libpcap standard, meaning that there are a variety of other tools that can be used to analyze the data it produces (such as windump).

The above five considerations were what made Snort a sure thing for this company, but it is important to analyze your own situation and chose what is right for you and your company.

[5] Trigeo Network Security. Contego Network Security Software
[6] The SANS Institute. SANS InfoSec Reading Room.

Without Further Ado…

Let's get started! The remainder of this paper will focus on installing and tuning Snort for the mission-critical server as described above. Installation, configuration, tuning the rule set, and log file monitoring and alerting options are still to come!

Installing Snort 2.1.1 as a Service in Windows 2000

The latest Win32 Binary installation for Snort can be obtained from the Snort website at http://www.snort.org/dl/binaries/win32/. At the time of the writing of this paper, the latest version is Snort 2.1.1. Once you have downloaded the self-executing installation file (it is a good idea to verify the MD5 checksum before installation), make sure that you are logged into the server with an account with Administrator privileges. You can then double-click the executable file and the installation will begin. There are a few additional things to consider when installing Snort for the purposes of this paper:

1) It is wise to put your Snort installation (or at least the logging directory) on a separate maintenance partition. This way, should the alert log file become incredibly large and fill the partition, you won't DoS your own production server. Usually, your maintenance partition is not the default C: drive, so this path will need to be changed during the installation.
2) One of the first questions you will be asked during installation is whether you plan to log to anything other than a MySQL or ODBC database. For the purposes of this paper, the default option is sufficient, but you will want to consider your own implementation.

Once the installation is complete (it does not require a reboot), there are a few other settings that need to be tweaked to make the installation more efficient for Windows. For starters, the default path statement on the server should be edited to include the path to the snort.exe file so that the full file path doesn't have to be entered every time Snort is executed. To do this in Windows 2000, right-click My Computer to open the System Properties window. Choose the Advanced tab, and select Environment Variables. In the System Variables field, double-click the Path option and add your path statement to this line (usually something like ;c:\snort\bin).

In addition to adjusting the path file, it is usually wise to run Snort as a service in a Windows environment. Since the whole idea behind monitoring traffic is to monitor it all the time, Snort will need to be running on the server 24/7. If Snort does not run as a service, then a user account will have to be logged into the server at all times; at best, not a good idea. To configure Snort to run as a service, open a command prompt and type the following:

```
snort /SERVICE /INSTALL [-options]
```

(Note: you may not want to perform this step until you have tested the options and rules files that you will use with Snort and are confident that they are functional! Recommended options will be discussed in the *Configuring Snort Options* section below. For more information on running Snort as a service, either type snort -? at the command line, or see the Snort FAQ, Section 5.5). [7]

[7] The Snort Core Team. The Snort FAQ.

Once the service has been installed, verify that it is running by using the Services applet in Administrative Tools. During testing, you may want to leave the service startup at Manual; however, once you are confident in your setup, the service should be configured to start automatically so that it will continue to run if the server reboots.

Another consideration for service configuration is the user account used to run it. Though the Local System account will suffice, it is usually wise to use a separate user account with a limited number of privileges to run the Snort service. This way, if you were unable to provide a dedicated maintenance partition for the Snort installation, you may want to create a local or domain-wide account and enable Disk Quotas for that user, to ensure that your log files do not fill your drive. For more information on enabling Disk Quotas in Windows 2000, see Managing Disk Quotas in Windows 2000. [8] As well, should a critical exploit be exposed for Snort, and your machine be compromised by it, the amount of damage done will be limited to the power of the user account you've created, as opposed to compromising the entire system (which the Local System account has the power to do).

Editing the snort.conf File

Since Snort was originally designed for Unix-based systems, there are a few additional steps required for a successful Windows installation, most of which take place in the snort.conf file. Most importantly, you will need to define the default variables and include the full path to all associated files within the snort.conf file. The following is an example of a snort.conf file, configured for a host-based installation on a server with the IP address of 10.10.1.1. Important entries are highlighted in bold, and comments have been truncated for the sake of space; all omitted data has been left at the default setting.

```
# Snort.Conf Rules file Example
var HOME_NET 10.10.1.1/32

var EXTERNAL_NET !$HOME_NET

#These setting will be server specific.  If a server runs that service,
variable should be set to HOME_NET; if not, then !HOME_NET.
# List of DNS servers on your network
var DNS_SERVERS $HOME_NET
#Server is a DNS server

# List of SMTP servers on your network
var SMTP_SERVERS !$HOME_NET

# List of web servers on your network
var HTTP_SERVERS !$HOME_NET

# List of sql servers on your network
var SQL_SERVERS !$HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS !$HOME_NET
```

[8] Microsoft Corporation. Managing Disk Quotas in Windows 2000.

```
# List of snmp servers on your network
var SNMP_SERVERS !$HOME_NET

# Path to your rules files (this can NOT be a relative path in Windows)
var RULE_PATH c:\snort\rules

# Once again, complete path must be specified in Windows
# Include classification & priority settings
include c:\snort\etc\classification.config

# Include reference systems
include c:\snort\etc\reference.config
```

Note: Once you have defined a rule set that you are comfortable with, you will want to add the extra step of copying the \snort\etc\snort.conf file, as well as the \snort\rules directory to each additional server installation to which that rule set applies.

Configuring Snort Options

There are many different switches associated with running Snort, and there are a few that are required in order to run a successful host-based implementation. The following is an example of the switches used in this installation (for more information on available options, type snort -? at the command line):

snort –A fast –bpc c:\snort\etc\snort.conf –l c:\snort\log

-A fast =	Use Fast logging mode; record packet headers only. Recommended for long-term installations of Snort.
-b =	Binary logging mode. A windump compatible binary capture file will be generated for only those packets that trigger an alert.
-p =	Disable promiscuous mode sniffing—turns off NIDS feature and only monitors traffic directed at the local interface. Very Important!
-c =	Path to the snort.conf rules file. Required in Windows implementations.
-l =	Path to Logging directory. Required in Windows implementations. Where the alert.ids and binary files will be stored.

That takes care of the options we will use. Be sure to include the –p option when starting Snort so that you don't turn your production server into a NIDS! Most (if not all) of your filters will be specified in the rules files found in \snort\rules directory, each of which must be included in the snort.conf file in order for Snort to consider them during packet processing (you may not want to include all rules, see the Tuning the Rule Set section below for more information).

Tuning the Rule Set

Beginning with the snort.conf file, there are several settings you will want to edit in order to reduce false positives and also in order to catch all suspicious traffic. In many cases when tuning the rules files, I left redundant but more specific rules that are included with the default installation enabled. For example, if I added a rule to alert on all traffic directed at port 21 (FTP), I kept the rules in the ftp.rules file enabled, so that should the attacker use a more specific form of attack I would be

alerted. This decision is entirely up to you; in some cases (telnet for example), I don't care what someone's doing at that port—if they're doing anything at all they're on my hit list. In these cases, I disabled the more specific rules to cut down on processing time. In general, you will definitely want to leave the more specific rules enabled for services that you are legitimately offering and need to monitor more closely (LDAP, DNS, etc) and consider disabling more specific rules on those services that should never be running on that server and just include a "catch-all" alert for that port in your local.rules file. In addition to adding your own rules, it is also a good idea to start off with all rules enabled in Snort (some come disabled by default, use your best judgment as to which could apply to your environment), and disable as experience and analysis permits.

False Positives:

When you first turn on the IDS, you will most likely find that you are being attacked from every direction! Not really; in most cases, your average Windows 2000 host can generate a lot of false positives, especially when dealing with ICMP. The Windows 2000 networking environment heavily relies on ICMP to function properly, and the following signatures are some that you may want to consider tuning and/or disabling.

L3 Retriever Ping (in the \snort\rules\icmp.rules file)
Signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP L3retriever
Ping"; content: "ABCDEFGHIJKLMNOPQRSTUVWABCDEFGHI"; itype: 8; icode: 0;
depth: 32; reference:arachnids,311; classtype:attempted-recon; sid:466;
rev:1;)
```

False Positive:
According to Whitehats, "This type of ICMP ping seems to be also generated by (plain) Win2K host talking to Win2K domain controllers." [9] And indeed it is! Unfortunately, I was unable to find (through my own research or through Google) a way of tuning this rule to ignore just the legitimate traffic. However, I am comfortable with commenting this rule out (placing a '#' sign in front of it), or with just restricting it to external IP addresses only, for two reasons: 1) According to Whitehats, this software is rare and this type of ping is rarely seen. I concur with this information, as I was unable to even find a copy of the software to test with; I believe that it has been discontinued since the original company was absorbed by Symantec. [10] 2) The L3 Scanner, when used at its default setting, also seems to trigger an HTTP scan alert, found in the web-misc.rules (WEB-MISC L3retriever HTTP Probe) which does not trigger any false positives that I am aware of, and can be left enabled.

ICMP Ping NMAP (in the \snort\rules\icmp.rules file)
Signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING NMAP";
dsize: 0; itype: 8; reference:arachnids,162; classtype:attempted-recon;
sid:469; rev:1;)
```

[9] Whitehats Network Security Resource. IDS311
[10] Whitehats Network Security Resource. IDS311—Research.

False Positive:
According to Whitehats, "This is only the default ping setting. Other ping software can emulate this signature." [11] If your network hosts are ever inclined to authenticate to a domain controller that is not in their own LAN, or if your network is slow, you may see false positives on this and the signature below. Windows 2000 workstations use what is referred to in the Microsoft world as Slow Link Detection.[12] Basically, a zero byte ping packet that matches the signature above is first sent out; if the response time is slower than a default value, a second larger packet is sent out which triggers the false positive below. Obviously, as security administrators we want to know about somebody using NMAP scans on the network, but I was unable to find any dissimilarity in these packets compared to that which NMAP sends. However, I did end up commenting this rule out, and this is why: NMAP, at its default setting and many others, does much more than just send one ping packet to map a network, or even to just fingerprint a host. In fact, when I ran it at its default against my IDS, even with this rule disabled, I still received 17 alerts, including HTTP, SNMP, Proxy, Finger, and FTP probes. In our case, some of our hosts occasionally authenticate to a domain controller located across a frame-relay link; when that occurs, many of these packets are generated. If you are not receiving an over-whelming number of these alerts, you may want to leave this rule enabled, though it is doubtful that you would miss an NMAP scan either way.

ICMP Large ICMP Packet (in the \snort\rules\icmp.rules file)
Signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Large ICMP
Packet"; dsize: >800; reference:arachnids,246; classtype:bad-unknown;
sid:499; rev:3;)
```

False Positive:
According to Whitehats, "The most common false-positive may be large ICMP Echo-Request packets (ping) that some programs may generate naturally." [13] In the case of a Windows network, this is the second packet involved in Slow Link Detection, which sends a 2 MB ICMP packet (containing the Microsoft logo) to the server in question. [12] Thankfully, this packet does have a distinct payload (thanks to that Microsoft logo), which can help in modifying this signature to remove false alerts, yet still alert on large ICMP packets. A sample packet capture of this traffic is included below, though it has been truncated for the sake of space:

```
04:57:45.624283 IP 10.10. 7.52 > 10.10.1.1: icmp 2056: echo request seq 45056
0x0000   4500 081c 1a8d 0000 7e01 76b9 c0a8 aa34        E.......~.v....4
0x0010   c0a8 7815 0800 09d5 0200 b000 ffd8 fffe        ..x.............
0x0020   0008 5741 4e47 3202 ffe0 0010 4a46 4946        ..WANG2.....JFIF
0x0030   0001 0101 0060 0060 0000 ffdb 0043 0010        .....`.`.....C..
0x0040   0b0c 0e0c 0a10 0e0d 0e12 1110 1318 281a        ..............(.
0x0050   1816 1618 3123 251d 283a 333d 3c39 3338        ....1#%.(:3=<938
0x0060   3740 485c 4e40 4457 4537 3850 6d51 575f        7@H\N@DWE78PmQW_
0x0070   6267 6867 3e4d 7179 7064 785c 6567 63ff        bghg>Mqypdx\egc.
0x0080   db00 4301 1112 1218 1518 2f1a 1a2f 6342        ..C......./../cB
0x0090   3842 6363 6363 6363 6363 6363 6363 6363        8Bccccccccccccc
0x00a0   6363 6363 6363 6363 6363 6363 6363 6363        ccccccccccccccc
0x00b0   6363 6363 6363 6363 6363 6363 6363 6363        ccccccccccccccc
0x00c0   6363 6363 ffc0 0011 0800 2600 9e03 0121        cccc......&....!
```

[11] Whitehats Network Security Resource. IDS162
[12] Microsoft Corporation. How a Slow Link is Detected for Processing User Profiles and Group Policy.
[13] Whitehats Network Security Resource. IDS246.

```
0x00d0   0002 1101 0311 01ff c400 1f00 0001 0501        ................
0x00e0   0101 0101 0100 0000 0000 0000 0001 0203        ................
0x00f0   0405 0607 0809 0a0b ffc4 00b5 1000 0201        ................
0x0100   0303 0204 0305 0504 0400 0001 7d01 0203        ............}...
0x0110   0004 1105 1221 3141 0613 5161 0722 7114        .....!1A..Qa."q.
0x0120   3281 91a1 0823 42b1 c115 52d1 f024 3362        2....#B...R..$3b
0x0130   7282 090a 1617 1819 1a25 2627 2829 2a34        r........%&'()*4
0x0140   3536 3738 393a 4344 4546 4748 494a 5354        56789:CDEFGHIJST
0x0150   5556 5758 595a 6364 6566 6768 696a 7374        UVWXYZcdefghijst
0x0160   7576 7778 797a 8384 8586 8788 898a 9293        uvwxyz.........
```

[snip]

As you can see, the packet contains a unique signature of "WANG2.....JFIF," which is the beginning of the Microsoft logo. The "pcre" option can be used in Snort to do a content match on this data and to not alert on the packet if this content is found. An example of the signature is given below:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Large ICMP,
NOT M$ Slow Link Det"; pcre:!"/WANG2/"; dsize: >800;
reference:arachnids,246; classtype:bad-unknown; sid:499; rev:3;)
```

I chose to use "pcre" instead of the "content" filter because of the content command's limitations when using a "not" option ('!'). In order to match a "not" content, the rule must first be able to match positively against something (for example, content: "JFIF"; content: !"WANG2") which seems to be computationally more expensive, since the packet payload would have to be examined twice. In the interest of keeping these rules as lightweight as possible the "pcre" option was used instead.

Note: Be sure to keep a close eye on your IDS installation when you first enable it for any period of time. If a lot of false positives are being generated, this can fill up a log file in a hurry!

Additions to the local.rules File:

Besides tuning for false positives, there are also several "catch-all" signatures that you may want to add to your local.rules file (a file included in the \snort\rules\ directory for including rules that are specific to the local environment). Though you can create your own separate rules files, the local.rules file is already empty and included in snort.conf by default. The following are some examples of "catch-all" rules that were added to my local.rules file. These rules will vary depending on the services you are running on the server.

```
# Local.rules file
# Last Modified 3-4-04

#Telnet Attempt
alert tcp any any -> $HOME_NET 23 (msg: "Local Telnet Attempt";)

#TFTP Attempt
alert udp any any -> any 69 (msg: "Local TFTP Attempt";)

#FTP Attempt--Active or Passive
alert tcp any any -> any 20 (msg: "FTP Port 20 Attempt";)
alert tcp any any -> any 21 (msg: "FTP Port 21 Attempt";)
```

```
#HTTP Probe
alert tcp any any -> $HOME_NET 80 (msg: "Local HTTP Probe";)
alert udp any any -> $HOME_NET 80 (msg: "Local HTTP Probe";)
alert tcp any any -> $HOME_NET 443 (msg: "Local HTTPS Probe";)
alert udp any any -> $HOME_NET 443 (msg: "Local HTTPS Probe";)

#SMTP Probe, only use if $HOME_NET is not an SMTP server!
alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg: "Attempted SMTP
Connection";)

#SNMP Traffic
alert udp any any -> any 161 (msg: "SNMP Traffic";)

#POP3 Traffic, only use if $HOME_NET is not a POP3 server!
alert tcp $EXTERNAL_NET any -> $HOME_NET 110 (msg: "POP3 Traffic";)

#Finger Attempt
alert tcp $EXTERNAL_NET any -> $HOME_NET 79 (msg: "Finger Attempt";)

#PC Anywhere attempt
alert tcp $EXTERNAL_NET any -> $HOME_NET 5631 (msg: "PC Anywhere TCP
5631 Attempt";)
alert tcp $EXTERNAL_NET any -> $HOME_NET 5632 (msg: "PC Anywhere TCP
5632 Attempt";)
alert udp $EXTERNAL_NET any -> $HOME_NET 5631 (msg: "PC Anywhere UDP
5631 Attempt";)
alert udp $EXTERNAL_NET any -> $HOME_NET 5632 (msg: "PC Anywhere UDP
5632 Attempt";)
```

Subtractions from the Default Rules Configuration:

As mentioned above, there may be some more specific rules that you will want to disable in the interest of keeping your rule set quick and speedy (assuming you have already covered that particular port in your local.rules file with a "catch-all" filter). As well, some rules may not apply to your environment (for example, if you're a Windows shop not running BIND, you may not wish to have BIND rules enabled...for now, I am choosing to leave mine enabled, as many widely available tools such as Nessus perform broad-based scans, and knowing what scans were performed can assist in fingerprinting the reconnaissance tool being used).

Telnet Rules:
Since none of my Windows 2000 servers have the telnet service enabled, nor should they, a telnet attempt to any of my servers is quite interesting to me. For that reason, I included a "catch-all" rule for all port 23 connections in my local.rules file (see *Additions to the local.rules file* above). Since any telnet connection will attract my attention, I chose to disable the extra telnet rules by commenting out the telnet.rules file in snort.conf and by commenting out the telnet rules found in \snort\rules\backdoor.rules. Once again, make the decision that is best for your environment; if you need to offer telnet on one of your servers, leave the more specific rules enabled.

Log Files: Monitoring and Alerting

Great! You've got Snort up and running! You've got lots of data! This is great…isn't it? In most cases, finding an effective way to analyze data and at the same time be immediately alerted of

important events (such as an active attack on your server) is one of the greatest challenges in the security world, especially in smaller organizations.

There are many different approaches to collecting log data, one of the most popular being to send the data to a SQL or Oracle database via syslog. For completely open-source users, this is a very viable approach. (For more information on this subject see the Snort Documentation pages at http://www.snort.org/docs/). [14]

There are also commercial options available that will accomplish this task, and tend to be a more practical (if more expensive) option in a Windows environment. Earlier I mentioned that in my organization we chose to purchase a commercial product called Contego, and that this product was capable of integrating Snort alerts into its own centralized logging console. This is accomplished via the SPOP, or Security Point of Presence, which is a small piece of software installed each desktop and workstation which "listens" for any important alerts (as defined by the security administrator), such as a virus detected on the workstation, or a security event on a server. When a flagged event is detected, the SPOP immediately reports that event back to a centralized console where it can then be prioritized and configured to alert an administrator if necessary.

To Configure Snort to Alert to the Trigeo Contego version 2.1 Console: [15]

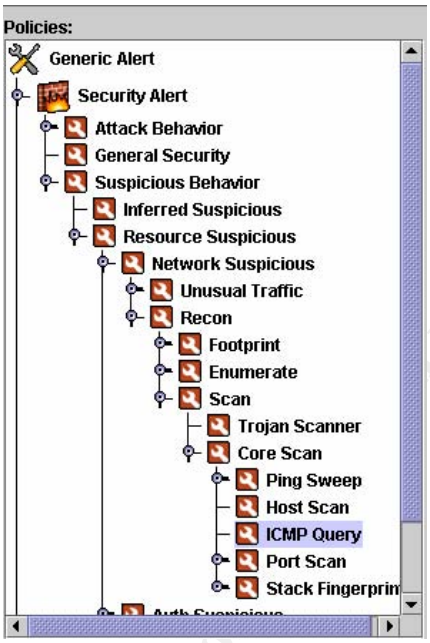
1. Install the SPOP on the server, via the Remote SPOP Installation utility included with your installation CD. (See Contego Installation and User Manual for more information.) [16]
2. Start the Snort service; be sure that Snort is configured to run in Fast alert mode (see *Configuring Snort Options* above for more information).
3. From within the centralized console, right click the server and select the Configure Tools option, then select Network-Based IDS.
4. Select the Snort option. Click New and give a Tool Alias name (i.e. SNORT). Make sure that the correct path to the alert.ids file is included under "Log File" (usually c:\snort\log\alert.ids).
5. Start the Tool; you should see a message that says the tool has started successfully. Click Next, Next, Finish.

The Snort installation on that server will now send all alerts to the Contego SPOP, which will forward them on to the Contego console. In order to configure alerting and email notification, Policy must be configured. Though it is beyond the scope of this paper to detail which events to alert on and to whom, the following is an example of how to configure an email notification to be sent to the Security Administrator whenever the SuperScan Echo alert is encountered.

1. Since a SuperScan echo is classified as an ICMP Query, that is where we will need to go to configure alerting. (Note: this will not cause the console to alert you on every ICMP Query, just those that are reported as alerts and/or suspicious behavior).
2. In the Contego window, select Manager, then Policy. Navigate to the ICMP Query child object (see Figure 2 below).

[14] Snort.org
[15] Pauls, Nicole
[16] Trigeo Network Security. Installation and User Manual.

Figure 2, ICMP Query Child Object



3. To the right of the navigation menu, under Policy Variables, Notification Configuration, place a check next to the Email option for your Security Administrator user. For more information on User Setup, see the Contego Installation and User Manual.[17]
4. Click OK to apply the changes. You may wish to test this configuration by running SuperScan against your server to see if the scan is detected. (A free copy of SuperScan can be downloaded from the Foundstone, Inc. web page.) [18]

Another benefit of using the Contego/SPOP notification system is that all alerts are passed to the console appliance via an encrypted SSL session, and the appliance itself if heavily secured, as well as running its own host-based firewall (iptables). This helps lessen the risk of having "the family jewels," namely the log file data from every host on your network, be compromised.

Logfile Rotation:

It is also worth noting that, at some point, the alert.ids and binary logging files generated by Snort will need to be rotated and archived, to avoid filling up unnecessary hard drive space on the server. When passing alerts to the Contego system, all log file data is recorded on the Contego appliance, so in our case it is unnecessary to keep the alert.ids file (though we do, mostly for audit purposes); I also like to archive the binary data, should further packet analysis be necessary. There are many ways to do this via automated script; however, for the sake of simplicity, I use a simple batch file on every server and the Windows Task Scheduler to make it run automatically once a month. When using the Windows Task Scheduler, you may want to run the task with the same user credentials that you run Snort with; this will make your life easier when auditing your network and performing

[17] Trigeo Network Security. Installation and User Manual.
[18] Foundstone, Inc.

user maintenance! (Note: I think you will find that, once you have tuned your rule set, you will receive only small number of alerts per server—just be sure you have not tuned it to the point of breaking it completely!)

In order to move or copy the alert.ids and binary log file data, the Snort service must first be stopped so that the files will no longer be in use. The following is an example batch script used for stopping the snort service, collecting log file data, deleting the old files, and then restarting the service. (Note: the alert.ids and binary files will be recreated automatically the next time Snort detects an alert).

```
#Batch script for processing log file data
#Stopping the Snort service
net stop “Snort”

#Sending log file data to centralized share, via a mapped drive
#Map Drive
net use Z: \\logserver\logshare
#Copy alert and binary files
copy c:\snort\log\alert.ids z:\%COMPUTERNAME%\
copy c:\snort\log\snort.log.* z:\%COMPUTERNAME%\

#Delete mapped drive
net use Z: /delete

#Delete old log files
del c:\snort\log\alert.ids
del c:\snort\log\snort.log.*

#Restart Snort Service
net start “Snort”
```

In this manner, the log files are cleaned off the servers and copied to a centralized share where they can be archived. Since all log files copy on the first day of the month, the \\logserver\logshare share only exists on the first day of the month and is then removed; the log files are burned to compact disc and removed from the server to reduce the chance of a security compromise. (Yet another benefit to using a dedicated domain user account to run your Snort services—it then becomes only user you have to give share permissions to, besides yourself!)

Conclusion:

In the never-ending effort to provide total network security, the greatest challenge is often ensuring that your trusted employees are not sabotaging their own networking environment, whether willfully or unwittingly. Though it would be ideal to be able to monitor and log every packet into and out of every workstation and act accordingly, in most cases it simply isn’t practical, especially for the small-medium sized business. In these situations, a compromise between total security and acceptable coverage must be reached; I believe that the combination of host-based IDS systems on mission critical servers, in tandem with a system for watching and monitoring only important

traffic on workstations (along with a NIDS at any Internet gateways and server segments) is a practical and acceptable way to accomplish this goal. This solution does present several drawbacks, the most obvious of which is the potential to miss a more sophisticated security invasion on a user workstation; however, the risk involved in the compromise of one workstation, or even more than one, is less than that involved in the compromise of a mission-critical server.

This paper focuses on implementing this security model specifically in a small-medium sized business with a Windows 2000 networking environment. Though the solutions presented here are scalable to a certain degree (especially through the use of scripts to automate the process, a topic not discussed in this paper in depth), this is not a practical enterprise solution. So, if you have smaller business with a limited budget and a limited number of resources (a common affliction amongst small to medium sized businesses), consider this to solution a cost-effective and practical way to aid in your quest for Windows network security!

References: White Paper

Andrews, Peter. "Internal Security: Getting a step ahead." Executive Technology Reports. 25 October 2001. URL: http://www-1.ibm.com/services/strategy/e_tek/internal_security.html.

Foundstone, Inc. "SuperScan v4.0." Free Tools. URL: http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/overview.htm

Microsoft Corporation. "How a Slow Link Is Detected for Processing User Profiles and Group Policy." Microsoft Knowledge Base Article – 227260. 3.0. 21 November 2003. URL: http://support.microsoft.com/?id=227260.

Microsoft Corporation. "Managing Disk Quotas in Windows 2000." Microsoft TechNet. URL: http://www.microsoft.com/technet/security/topics/issues/w2kccadm/routine/w2kadm36.mspx

Pauls, Nicole. "Snort Question." Email to Nicole Pauls, Trigeo Support. 13 January 2004.

Snort.org. Documentation—Setup Guides. 15 March 2004. URL: http://www.snort.org/docs/.

TechRepublic. "A multilayered strategy helps neutralize internal security threats." 1 July 2002. URL: http://techrepublic.com.com/5100-6313-1051506.html.

The SANS Institute. SANS InfoSec Reading Room. URL: http://www.sans.org/rr/.

The SANS Institute. "Windows 2000 Issues." SANS InfoSec Reading Room. 15 March 2004. URL: http://www.sans.org/rr/catindex.php?cat_id=66.

The Snort Core Team. "The Snort FAQ." 4 September 2003. URL: http://www.snort.org/docs/FAQ.txt.

Trigeo Network Security. "Contego Network Security Software." Products. URL: http://www.trigeo.com/products.php

Trigeo Network Security. Installation and User Manual. 93-102.

Waveset. "Internal Threats are Real." URL: http://www.waveset.com/Features/inside_threat.html.

Whitehats Network Security Resource. "IDS162 'DOS-LARGE-ICMP'." arachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/info/IDS246.

Whitehats Network Security Resource. "IDS162 'PING-NMAP-ICMP'." arachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/info/IDS162.

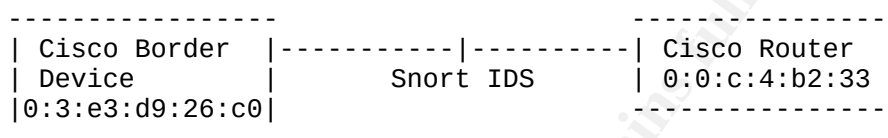
Whitehats Network Security Resource. "IDS311 'PING-SCANNER-L3RETRIEVER'." arachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/info/IDS311.

Practical Detects

BACKDOOR Q Trojan, Practical Detect #1

1. Source of Trace:
   This capture file, 2002.10.17, was obtained from the raw logs at www.incidents.org/logs/raw and was posted Monday, December 2, 2002.

   Network Architecture:
   Though the exact network architecture is unknown, it can be surmised that the layout is similar to the following:



   This assumption is made based on the fact that all traffic originates from two MAC addresses, both of which are owned by Cisco Systems, according to http://www.coffer.com/mac_find/, and Ethereal version 0.9.9. A sample packet, captured with the following windump command, illustrates this:

```
windump -nr 2002.10.17 -e host 255.255.255.255
```

   -n = Do not resolve host names
   -r = Read from binary follow
   -e = Display Link layer information (MAC addresses)

```
15:00:24.516507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: IP
255.255.255.255.31337 > 170.129.153.131.515: R 0:3(3) ack 0 win 0
```

   It can also be assumed that the Cisco device with the physical address of 0:3:e3:d9:26:c0 is a border router/firewall, since all traffic originating from this address is from public IP address space, and is destined for the internal address space of 170.129.0.0/16 (a standard Class B network structure).

2. Detect was generated by:
   Snort Intrusion Detection System version 2.0.2, running in full alert mode, with all rule files included. The following is an example of the alerts that were logged:

```
[**] [1:184:3] BACKDOOR Q access [**] [Classification: Misc
activity] [Priority: 3] 11/17-15:00:24.516507
255.255.255.255:31337 -> 170.129.153.131:515
TCP TTL:14 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]
```

Which was triggered by the following rule in the backdoor.rul file (the HOME_NET variable is set to "any" in the snort.conf file):

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q
access"; flags:A+; dsize: >1;  reference:arachnids,203; sid:184;
classtype:misc-activity; rev:3;)
```

In reference to the sample TCP packet above, this alert was triggered based on the fact that the source address matched the 255.255.255.0/24 requirement (since 255.255.255.255 is on the 255.255.255.0/24 subnet), as well as had the ACK flag set (and, in this case, the RST flag as well). In addition, the payload size (dsize) was greater than 1 byte. In this particular case, the ports used (31337 and 515) were not considered in the rule definition (defined only as "any").

3. Probability the source address was spoofed:
   Very high. The 255.255.255.255 address space is reserved for broadcast traffic, and should never be seen as a source address in any legitimate packet (RFC 919). [19]

4. Description of Attack:
   This attack (referenced at http://www.whitehats.com/info/IDS203, CVE# CAN-1999-0660) is a backdoor attack mainly targeted at Unix systems. The Q daemon, written by Mixter (http://mixter.void.ru), is designed to be a covert communication channel between a client (attacker) and server (victim) that allows the attacker to execute remote commands, shell processes, and port redirections. Communication between the client and an already infected "server" are initiated by a control packet that is sent to the TCP layer of the said server (UDP and ICMP protocols are also supported, though not present in this capture).

   An initial control packet is sent to the server, designed with the single purpose of executing a command on the server (i.e. opening a shell process on a certain port specified by the attacker); neither a response packet nor an established TCP session are required in order for the control packet to be successful. However, in order for this packet to be effective, the Q daemon must already be installed (with root privileges) on the victim machine by some other means (i.e. from an attacker's rootkit, or by email or IRC).

   The packets below are a representative sample of the Q Backdoor traffic being discussed.

```
(SNORT OUTPUT)
[**] [1:184:3] BACKDOOR Q access [**] [Classification: Misc
activity] [Priority: 3] 11/17-15:00:24.516507
255.255.255.255:31337 -> 170.129.153.131:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] [1:184:3] BACKDOOR Q access [**] [Classification: Misc
activity] [Priority: 3] 11/17-05:53:43.966507
255.255.255.255:31337 -> 170.129.153.221:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
```

[19] Mogul, Jeffrey

```
[**] [1:184:3] BACKDOOR Q access [**]Classification: Misc
activity] [Priority: 3] 11/17-06:18:14.206507
255.255.255.255:31337 -> 170.129.129.38:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20

(WINDUMP OUTPUT)
15:00:24.516507 IP (tos 0x0, ttl 14, id 0, len 43)
255.255.255.255.31337 > 170.129.153.131.515: R [tcp sum ok]
0:3(3) ack 0 win 0 [RST cko]
4500 002b 0000 0000 0e06 68c9 ffff ffff        E..+......h.....
aa81 9983 7a69 0203 0000 0000 0000 0000        ....zi..........
5014 0000 1cf1 0000 636b 6f00 0000             P.......cko...

05:53:43.966507 IP (tos 0x0, ttl 14, id 0, len 43)
255.255.255.255.31337 > 170.129.153.221.515: R [tcp sum ok]
0:3(3) ack 0 win 0 [RST cko]
4500 002b 0000 0000 0e06 686f ffff ffff        E..+......ho....
aa81 99dd 7a69 0203 0000 0000 0000 0000        ....zi..........
5014 0000 1c97 0000 636b 6f00 0000             P.......cko...

06:18:14.206507 IP (tos 0x0, ttl 14, id 0, len 43)
255.255.255.255.31337 > 170.129.129.38.515: R [tcp sum ok] 0:3(3)
ack 0 win 0 [RST cko]
4500 002b 0000 0000 0e06 8126 ffff ffff        E..+.......&....
aa81 8126 7a69 0203 0000 0000 0000 0000        ...&zi..........
5014 0000 354e 0000 636b 6f00 0000             P...5N..cko...
```

These packets are characteristic of the Q trojan for the following reasons:

A) As mentioned above, the packets' source IP addresses are spoofed, originating from a static source port, going to an apparently randomly generated destination IP with a static destination port. (Destination IP addresses are assumed to be randomly generated as the addresses don't appear to follow any standard subnet boundaries or incremental values, but are targeted at various IP addresses on the 179.129.x.x/16 internal subnet). Note: Though this particular Snort rule is designed to trigger on source addresses of the 255.255.255.0/24 value, the Q Trojan does not exclusively use this address as a source IP value.

B) The IP ID value is static; in this case, a value of 0 in every packet, which is further evidence of packet crafting.

C) The packet is purposely mis-crafted to prevent any response from occurring, as evidenced by the fact that the RST and ACK flags are simultaneously set. When the victim's TCP stack receives this packet without an established session, it should be silently dropped, according to RFC 793. [20] This is beneficial to the attacker, as it allows him to spoof the source IP address and mask his true identity.

[20] Information of Sciences Institute

5. Attack Mechanism:

   The packets captured are assumed to be the initial "control packet" being sent out to various IP addresses to try and find a machine infected with the Q daemon. The payload of the packets ("cko") offers little clue as to the actual command being sent to the daemon; in fact, this may be the indication of a broken tool, or an attempt to exploit another weakness. Vulnerabilities do exist for the LPD (Line Printer Daemon), which commonly listens on port 515; however, these packets do not seem to be effective attacks for any of the well-known buffer overflow vulnerabilities the LPD is susceptible to. [21] This is evidenced by the fact that these packets have been purposely mis-crafted (with the RST and ACK flags set) and will be dropped by any properly configured TCP stack, never to be passed on to the LPD service.

   Since the packets are so widespread and weakly disguised, it can probably be assumed that the target machines are not already known to be infected, and the attacker is fishing for a victim. Whether or not the attacker found a victim will remain a mystery, as this capture does not contain any potential response traffic from the targeted hosts (due to the fact that the capture files were saved in Snort binary logging mode, so only the packets that triggered the alert were captured). The attacker does demonstrate some intimacy with network, however, since they are obviously aware that the border router/firewall is allowing incoming port 515 traffic (assuming that the device is at least somewhat hardened, and not simply allowing all traffic to enter the network).

   As to the version of Q daemon used, it is safe to assume that the packets captured were generated by a pre-2.0 version. This assumption is based on the fact that the packet is in clear-text (later versions support encryption), and the fact that the source IP address was statically assigned to 255.255.255.255 (this was also only possible in pre-2.0 versions). [22] However, these assumptions are only valid if the original Mixter code was used, unaltered. It is also interesting to note that different client/server versions of the Q Trojan are not cross-compatible by default, which further narrows the chances of the attacker finding a valid listening Q daemon, without having prior knowledge of an infected machine.

6. Correlations:
   As this capture file is publicly available, many different sources have analyzed the file and reached various conclusions. The following SANS submissions were noted while researching this detect: Les Gordon (http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc) and Mike Shannon (http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00144.html). In addition to his own GCIA submission, Les Gordon also wrote a SANS Intrusion Detection FAQ on the Q Trojan, found at (http://www.sans.org/resources/idfaq/qtrojan.php).

[21] CERT/CC
[22] Gordon, Les

The CVE entry for this attack, CAN-1999-0660, is still under review (and has been since 1999) and provides only a general description, recognizing that the Q Trojan is a backdoor vulnerability. [23]

7. Evidence of Active Targeting:
   There doesn't appear to be any evidence of active targeting in this capture file. Of the 43 different IP addresses affected, no two addresses were attacked twice, nor do the addresses chosen appear to follow any standard subnet schemes or incremental values. The following packet sample illustrates this:

```
16:58:00.016507 IP 255.255.255.255.31337 > 170.129.214.158.515
18:49:41.436507 IP 255.255.255.255.31337 > 170.129.122.35.515
19:54:27.286507 IP 255.255.255.255.31337 > 170.129.106.86.515
20:54:38.656507 IP 255.255.255.255.31337 > 170.129.52.209.515
21:10:57.776507 IP 255.255.255.255.31337 > 170.129.186.20.515
```

8. Severity:

   * Severity = 4 *

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

   Criticality = 4. Being that the purpose of this network is virtually unknown, criticality has been rated at a four. It seems unlikely that this subnet is a Screened Subnet or Demilitarized Zone, as it appears to contain so many hosts (Class B). Therefore, it is probably an internal LAN or WAN, indicating that it may contain any range of services from DNS, Mail, domain controllers, and almost certainly a web server, judging by other packets directed at port 80 in this capture (i.e. a much wider range of services than your average DMZ). I think it is safe to assume that this is a large network that contains servers that are critical to its own functionality, if not also to this company's customers.

   Lethality: = 5. If this attack succeeded as planned, and the attacker was able to find a compromised Q server, that attacker could then execute virtually any code of their choice, with root privileges, on that box. If that box were to be a mission-critical server (such as a file or database server, or a mail server), the potential damage is high, ranging from a simple file transfer to complete software destruction.

   System Countermeasures = 3. Given that absolutely nothing is known about the hosts involved in this attack (fingerprinting tools such as p0f are unreliable when used on crafted packets), I have rated System Countermeasures at 3. Since even the attacker seems unsure of the existence of an infected system, it is quite possible that none exists and all machines have been patched and secured. However, should the attacker find a box that has been compromised, it is likely that the machine was not well secured, which was what made it susceptible to the initial compromise.

[23] Common Vulnerabilities and Exposures

Network Countermeasures = 2. Though there is apparently some sort of Cisco border router/firewall in place on this network, it is allowing port 515 traffic, as well as erroneous source addresses, into the network. Since an infected machine could be damaged with one packet, issuing a single command, the most important line of defense to protect against the Q Trojan is the perimeter configuration; not only to protect against the initial compromise, but also to prevent the machine from receiving control packets. Since the border device is allowing the control packet to enter the network, it is not well configured to defend against this type of an attack. However, this may be one of only a few ports that the border device is allowing through (exact configuration is unknown, however, six unique open ports were present in this capture file alone), thus limiting the range of communication channels available to a potential attacker and requiring more reconnaissance on his part, which increases the chances of detecting the attack before it occurs.

9. Defensive Recommendations:

Since it is unknown whether or not any internal machines are infected with the Q Trojan and need to be cleaned and patched, I will focus this section on perimeter defense mechanisms. However, it should be mentioned that a system for regularly identifying and applying patches, service packs, and anti-virus updates should be designed and implemented on the network, in order to keep internal machines as protected as possible and thus decrease their risk of initial infection.

As mentioned above, the border device is the most important consideration in securing this network from the Q Trojan attack. Several blaring misconfigurations are immediately obvious:

1) It is allowing port 515 traffic to pass into the network. Considering that this is a port commonly associated with printer services, it seems unlikely that it is necessary to allow this traffic in. Also, an audit should be performed on this device to ensure that it is not allowing any other unnecessary port traffic through; only those ports that are absolutely necessary to the functionality of the network should be open. It should be noted that at least 6 unique open ports were found in the traffic present in the capture file, including ports 80 (HTTP), 8080 (proxy), 1080 (proxy) 3128 (RingZero, Squid-HTTP), and 139 (NetBIOS)—it is likely that at least three of these ports (3128, 139, and 515) should not be open on the external interface. Since this is a Cisco device, allowable traffic can be easily configured through the use of an Access List (or several) on the external interface. If services such as root DNS servers, external web or mail servers, or credit card services are running on the network, these servers should be moved to a screened subnet or DMZ, in order to prevent having to allow this "high-risk" traffic into the internal network.

2) The device is allowing erroneous source addresses (i.e. 255.255.255.255) to pass into the network. Private, non-routable addresses (192.168…, 10.10…, etc), broadcast addresses, and possibly InterNic reserved addresses, should all be blocked from entering the network from the outside. This is just another security measure that will help stop any number of crafted and/or illegitimate packets from entering the network.

It should also be verified that the border device is properly patched and updated, as well as part of a regular audit and vulnerability assessment program to ensure continued security.

In addition to the above mentioned security enhancements, the default Snort IDS signature that triggered this alert is also a point for improvement. The default signature is restricted to the 255.255.255.0/24 address space for the source address of the attacking packet. However, the Q Trojan can use any source address to initiate an attack; therefore if the IDS used to capture this alert is running a default configuration, it could potentially miss malicious Q packets. Les Gordon, in his Intrusion Detection FAQ addresses this issue and gives several suggestions for improving the default Snort signature. [24]

10. Multiple choice test question:
When the following packet is received by the RFC-compliant target host, what will occur?

```
05:53:43.966507 IP (tos 0x0, ttl 14, id 0, len 43)
255.255.255.255.31337 > 170.129.153.221.515: R [tcp sum ok]
0:3(3) ack 0 win 0 [RST cko]
```

A) A Q daemon cko control packet will be sent back to the source address
B) If port 515 is open, a RST packet will be sent to 255.255.255.255, causing a DoS attack
C) If port 515 is closed, a RST packet will be sent to 255.255.255.255, causing a DoS attack
D) The packet will be silently dropped, and may or may not initiate a Q daemon session.

Answer: D

Incidents.org Feedback for Practical Detect #1:

The top three questions were submitted by Don Murdoch, via email:

Question: "djm - how many did you get" (Referring to Backdoor Q Trojan Alerts)
"how many did you get from given source addresses?"

Response: In this case, there was only one relevant source address, 255.255.255.255. A total of 43 alerts were detected. This information was obtained by using the following command:

grep BACKDOOR c:\snort\log\alert.ids

Question: "djm - what can be done to monitor for victims?"

Response: This is a tough one with this Trojan, since it doesn't pick any one port or service. However, I would think that a HIDS, or even just software for verifying file integrity would go a long way in helping alert the admin to a compromised box.

[24] Gordon, Les

Question: "explain your answer" (Referring to Multiple Choice question)

Response: Answer: D. According to RFC 793, any time a packet with the RST and ACK flags set is received by the target TCP stack (especially without a previously established session), the packet will be silently dropped. However, due to the nature of the Q Trojan, the packet does not have to be successfully processed by the target in order to start a Q session--whether or not this happens would depend on whether or not the Q daemon already existed on the machine and was listening on port 515.

DNS version.bind Request, Practical Detect #2

1. Source of Trace:
   This capture file, 2002.4.16, was obtained from the raw logs at www.incidents.org/logs/raw and was posted June 4, 2002.

   Network Architecture:
   Though the exact network architecture is unknown, it can be surmised that the layout is similar to the following:

```
-----------------                    ----------------
| Cisco Border  |-----------|----------| Cisco Router  |
| Device        |       Snort IDS      | 0:0:c:4:b2:33 |
|0:3:e3:d9:26:c0|                      ----------------
-----------------
```

   This assumption is made based on the fact that all traffic originates from two MAC addresses, both of which are owned by Cisco Systems, according to http://www.coffer.com/mac_find/, and Ethereal version 0.9.9. A sample packet, captured with the following windump command, illustrates this:

```
windump -nr 2002.4.16 –e dst port 53
```

   -n = Do not resolve hostnames
   -r = Read from binary file
   -e = Display link layer information (MAC addresses)

```
16:57:33.244488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 72: IP
203.155.237.173.4401 > 78.37.86.123.53:  4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)
```

   It can also be assumed that the Cisco device with the physical address of 0:3:e3:d9:26:c0 is a border router/firewall, since all traffic originating from this address is from public IP address space, and is destined for the internal address space of 78.37.0.0/16 (a standard Class B network structure).

2. Detect was generated by:
Snort Intrusion Detection System version 2.1.1, running in full alert mode, with all rules included in snort.conf. Checksums were ignored in the file since the internal IP addresses and checksums were obfuscated before posting (www.incidents.org/logs/raw/readme). Snort was executed using the following command:

```
snort –A full –k none –c c:\snort\etc\snort.conf –l c:\snort\log
–r c:\snort\2002.4.16
```

-A = Define alert mode (full, fast, console, or none
-k none = Ignore checksums
-c = Path to rule file
-l = Logging directory
-r = Read from binary file

A total of 45 alerts were logged relating to this detect. The following is an example of the alerts that were logged:

```
[**] [1:1616:4] DNS named version attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/15-16:57:33.244488 203.155.237.173:4401 -> 78.37.86.123:53
UDP TTL:44 TOS:0x0 ID:23503 IpLen:20 DgmLen:58
Len: 30
[Xref => http://www.whitehats.com/info/IDS278][Xref =>
http://cgi.nessus.org/plugins/dump.php3?id=10028]
```

Which was triggered by the following rule in the \snort\rules\dns.rule file (the EXTERNAL_NET and HOME_NET variables are both set to "any" in the snort.conf file):

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named
version attempt"; content:"|07|version"; nocase; offset:12;
content:"|04|bind"; nocase; offset: 12; reference:nessus,10028;
reference:arachnids,278; classtype:attempted-recon; sid:1616;
rev:4;)
```

In reference to the sample packet above, this alert was triggered based on the following criteria:

A) It is a UDP packet destined for port 53 on our Home Network.
B) It contains the payload content "|07|version" and "|04|bind", offset at least 12 bytes into the packet. (The | symbol indicates raw, or hexadecimal, payload data). [25] Note: this content filter is not case sensitive, as noted by the "nocase" option.

3. Probability the source address was spoofed:
Medium. This type of attack is considered a reconnaissance operation, and usually precedes an actual attack. In order for this query to be effective, the potential attacker must receive the response to their question (in this case, the question being what version of BIND the DNS server is running) and then choose an appropriate attack method.

[25] The Snort Project (Snort Users Manual, section 2.5.1)

However, it should be noted that UDP is not connection-oriented, and therefore is considered unreliable and more susceptible to spoofing. It would be beneficial to the attacker to mask his true identity; one way he could do this and still receive a response to his query is to only spoof part of the IP packets, thus creating "decoys" to help confuse the situation. One or two of the IP addresses would valid addresses he owns, while the rest would be widespread in order to help mask his identity.

Although fingerprinting tools such as p0f are ineffective against UDP packets (because UDP is connection-less), there are several other hints that at least a few of these packets are not spoofed. First, the IP ID numbers are incrementing with each packet from one of the source IP addresses. This points towards packets that are genuinely coming from the same host, as the IP ID value increments each time a host sends a new packet; therefore, packets that are originating from a single host in a short period of time should have a gradually incrementing IP ID value. (Whether that value is fixed or random depends upon the OS). As well, TTL values are consistent on packets from the same source address, indicating that the point of origin is remaining constant (it also seems likely that the originating hosts are running a version of *nix, which is the most common OS with a starting TTL of 64—the closest power of 2 value to 44). [26] (See sample packets below.)

```
05:52:45.274488 IP (tos 0x0, ttl 44, id 2101, len 58)
217.131.191.70.1834 > 78.37.178.35.53
07:20:50.354488 IP (tos 0x0, ttl 44, id 39657, len 58)
217.131.191.70.3905 > 78.37.174.14.53
10:43:10.364488 IP (tos 0x0, ttl 44, id 39745, len 58)
217.131.191.70.4197 > 78.37.225.167.53
10:54:11.424488 IP (tos 0x0, ttl 44, id 64816, len 58)
217.131.191.70.3930 > 78.37.0.83.53
```

It is interesting to note that not all of the packets from the same source IP addresses follow this pattern. As you will see in the next sample below, most of the IP ID numbers are very sporadically spread out. It is still probable that the packets are part of a scanning tool or worm, but I believe that the attacker desires a response to the query, as there are not enough packets noted here to constitute any sort of DoS attack, nor are the payloads of these packets large enough to make an effective DoS attack.

4. Description of Attack:

As mentioned above, these packets are less evidence of a specific attack, and more evidence of an attack to come (reconnaissance). A UDP packet containing a version.bind request is sent to a target host. Assuming that the target host has a DNS server running BIND and listening on port 53, it will reply to the source address with the version of BIND it is running. Once this information has been obtained, the attacker would craft an attack based on the vulnerabilities of that version. BIND is a widespread standard for hostname resolution, especially among Unix-based hosts, and there are many different vulnerabilities for many different versions. (CVE alone lists 34 different BIND vulnerabilities). [27]

[26] Zalewski, Michal
[27] Common Vulnerabilities and Exposures

5. Attack Mechanism:

The following is a representative packet sample of the logged traffic (checksum data truncated to save space):

Sample #1, Source IP Address: 203.155.237.173:

```
16:57:33.244488 IP (tos 0x0, ttl 44, id 23503, len 58)
203.155.237.173.4401 > 78.37.86.123.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

17:16:13.574488 IP (tos 0x0, ttl 44, id 42942, len 58)
203.155.237.173.2205 > 78.37.49.231.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

17:32:51.604488 IP (tos 0x0, ttl 44, id 9832, len 58)
203.155.237.173.1585 > 78.37.24.41.53: 4660
 [b2&3=0x80] TXT CHAOS? version.bind. (30)

17:41:54.684488 IP (tos 0x0, ttl 44, id 21486, len 58)
203.155.237.173.4135 > 78.37.164.210.53: 4
660 [b2&3=0x80] TXT CHAOS? version.bind. (30)

18:43:15.784488 IP (tos 0x0, ttl 44, id 53410, len 58)
203.155.237.173.3688 > 78.37.230.50.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)
```

Sample #2, Source IP Address: 217.131.191.70

```
05:52:45.274488 IP (tos 0x0, ttl 44, id 2101, len 58)
217.131.191.70.1834 > 78.37.178.35.53: 4660
 [b2&3=0x80] TXT CHAOS? version.bind. (30)

07:20:50.354488 IP (tos 0x0, ttl 44, id 39657, len 58)
217.131.191.70.3905 > 78.37.174.14.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

10:43:10.364488 IP (tos 0x0, ttl 44, id 39745, len 58)
217.131.191.70.4197 > 78.37.225.167.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

10:54:11.424488 IP (tos 0x0, ttl 44, id 64816, len 58)
217.131.191.70.3930 > 78.37.0.83.53: 4660[b2&3=0x80] TXT CHAOS?
version.bind. (30)
```

As demonstrated in the first packet sample above (of which there were a total of 13 packets over 7 hours recorded), the TTL values on these packets are remaining constant, and the IP ID values are unique, but random, not sequential. As well, the DNS ID number remains at a constant 4660, indicating the use of some type of automated scanning tool, the source(s) of which are similar in OS and hop count (relative to the target). The majority of the packets collected by this capture file are similar to Sample #1, and share the same TTL and random IP ID values. As well, the same IP address is never targeted twice. I

believe these packets to be spoofed, mostly based on the fact that the IP ID values are not consistent with traffic originating from one source machine.
However, there are two source IP addresses in particular that caught my eye when examining this data. The first, which is referenced by the second packet sample above (source IP 217.131.191.70) seems unique in the fact that its IP ID values increment sequentially as time passes, indicating that the packets are genuinely originating from the same source machine. A total of 4 packets over a period of 5 hours were captured from this host. A Whois lookup of this network indicated that the 217.131.0.0 – 217.131.255.255 IP address space belongs to the Ripe Network Coordination Center, which further allocated it to Superonline.net (this information was obtained from the Ripe Whois Database at www.ripe.net). In addition to this packet sample, another set of packets was obtained from a host on the same 217.131.x.x subnet: source host 217.131.173.179. Though these packets contained the same random IP ID values as the other spoofed packets, the TTL was 3 hops closer than the other (TTL = 47—see Sample #3 below).

Sample #3, Source IP Address: 217.131.173.179

```
22:51:01.534488 IP (tos 0x0, ttl 47, id 40120, len 58)
217.131.173.179.3627 > 78.37.136.3.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

22:57:15.024488 IP (tos 0x0, ttl 47, id 49241, len 58)
217.131.173.179.2290 > 78.37.129.135.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

00:36:00.264488 IP (tos 0x0, ttl 47, id 6345, len 58)
217.131.173.179.3871 > 78.37.36.217.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

01:32:57.634488 IP (tos 0x0, ttl 47, id 1373, len 58)
217.131.173.179.4646 > 78.37.95.34.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

02:00:31.214488 IP (tos 0x0, ttl 47, id 63486, len 58)
217.131.173.179.2073 > 78.37.85.22.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)

02:27:49.374488 IP (tos 0x0, ttl 47, id 56244, len 58)
217.131.173.179.3206 > 78.37.82.208.53: 4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)
```

I find it to be more than just coincidence that the only two source addresses with unique characteristics in this file are located on the same subnet, relatively close to one another (without knowing the exact subnet structure, I cannot be sure of physical proximity). I believe that the true attacker resides on the 217.131.x.x subnet; he may own other boxes in other subnets that he used to mask his attack, but I think that it is more likely that he used several machines on his own subnet to launch a distributed reconnaissance attack (mostly because of the consistent TTL values, which are more difficult to spoof when originating from different subnets), all of which used spoofed IP addresses *except* for the 217.131.191.70 host. In this way he could mask his true identity in a flood of packets

seemingly from different networks, yet still use a single valid IP address to get the BIND version information he needs. As for the unique TTL value of the 217.131.173.179 host, I can only assume that this was yet another box owned by the attacker (because of the subnet similarities), just physically closer to our target.

This data is obviously not conclusive, and full packet captures of traffic associated with the suspect IP addresses would be needed in order to determine which, if any, IP addresses followed these probes with a related DNS BIND attack, or performed further probes.

As for the tool used to generate the packets, it is difficult to say exactly, as there are many tools that will recreate the packets that were captured in this file. As seen in the Snort web reference, Nessus plugin number 10028 will duplicate packets such as these by enabling the "General, Determine which version of BIND name daemon is running" plugin (when run with Linux POSIX GUI client). As well, nslookup is capable of generating these packets by using the following command:

```
nslookup –q=TXT –class=CHAOS version.bind. <target host>
```

In addition to Nessus and nslookup, NMap version 3.48 will also generate a version.bind TXT CHAOS request when fingerprinting a host for services. NMap includes a switch option (-D) to use decoy IP addresses when running the scan, which sends the data from multiple IP addresses (as well as the real one) and allows an attacker to mask their identity. Although the default –sV scan covers many different ports other than just DNS (of which there is no evidence of in this packet capture), it is possible to modify the nmap-service-probes file used with NMap to just target port 53. (See example command below.)

```
nmap –sV –D ip1,ip2,ip3 <target host>
```

As noted by Beth Binde in her own GCIA submission, there is also a worm called the Lion worm that will generate a version.bind request and target Linux hosts. [28] This worm spreads by first performing a random port scan to random class B network addresses in order to find a BIND vulnerability and gain root access. I was only able to find limited detail on the worm itself, but nowhere was it indicated that the worm was capable of spoofing the IP address of the source machine; for this reason, I ruled out the possibility of this worm because the of the randomness of the IP ID fields in most of the packets, strongly indicating that they have been spoofed. As well, after looking at a packet capture of the Lion attack sequence at http://seclists.org/lists/incidents/2001/Mar/0274.html, it seems that the version.bind request used by the worm is inconsistent with the packets in this capture.

Based on the evidence above, I would conclude that this was not the work of a worm, or even many different random attacking hosts, but the work of a sophisticated attacker attempting to mask his identity.

[28] Binde, Beth

6. Correlations:
An ARIN Whois lookup on the source IP addresses involved in this attack revealed that all of the attacks originated from one of two ISPs:

A) Asia Pacific Network Information Center
202.0.0.0 – 203.255.255.255 (31 packets recorded)

B) Ripe Network Coordination Center
217.0.0.0 – 217.255.255.255 (14 packets recorded)

There were a total of 10 unique source IP addresses, as reported by SnortSnarf.

This attack is categorized as CVE-1999-0009 in the CVE database, and is listed as "Inverse query buffer overflow in BIND 4.9 and BIND 8 Releases." [29] More information about other BIND attacks can also be found by doing a search for keyword BIND in the CVE database, and a list of vulnerable version numbers is located at http://www.securityfocus.com/bid/134.

More information on the snort signature and packet captures can be found in the Whitehats Arachnids database, IDS278 (http://www.whitehats.com/info/IDS278).

More information on the specific Nessus plugin capable of performing this attack can be found at http://cgi.nessus.org/plugins/dump.php3?id=10028.

The p0f version 1.8.2 fingerprint file was used as a basis for determining the likely OS of the attacking hosts.

In addition to the above sources, the following GCIA submissions and their associated responses were also used in this analysis:
Beth Binde: http://www.giac.org/practical/GCIA/Beth_Binde_GCIA.pdf

Todd Williams:
http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00102.html

7. Evidence of Active Targeting:
At first glance it doesn't appear as though any specific IP addresses were targeted in the attack, mostly because of the lack of any one IP being targeted twice. The attack most definitely targets DNS servers, but whether or not the selected IP addresses are actually listening with BIND on port 53 will remain a mystery, without a complete packet capture. Judging by the randomness of the IP addresses targeted (the addresses don't follow any common incremental values or subnet boundaries), it would seem that the attacker does not already know which targets are listening on port 53.

However, if the attacker is attempting to mask his identity by sending a barrage of packets to the network, it is then possible that the attacker already knows that certain servers are

[29] Common Vulnerabilities and Exposures

running BIND; these would be the addresses that were targeted by the 217.131.191.70 source machine (78.37.178.35, 78.37.174.14, 78.37.225.167, 78.37.0.83). There were, however, no other packets in the capture file to or from these hosts to confirm or disprove this theory.

8. Severity:

* Severity = 3 *

Severity = (Criticality [4] + Lethality [5]) –
(System Countermeasures [3] + Network Countermeasures [3])

Criticality = 4. Without knowing the business-model and purpose of this company, it is difficult to determine the true value of DNS to this network. However, it is safe to assume that at the very least internal availability for file/application servers and/or any internal web servers would most likely be lost should name resolution become unavailable. There is also a chance that this company's business depends on their customers being able to access their web server; if the company is hosting their own (and only) primary and secondary name servers for this web access, their removal becomes more critical.

Lethality = 5. Although the packets we are seeing are not an actual attack, if an attacker were to find a vulnerable version of BIND and exploit it, it is likely that they would gain root access to that server. After that, the possibilities for mayhem and destruction are endless.

System Countermeasures = 3. Since no potential response packets were captured, this is nearly impossible to determine. It is possible that even if the host were running BIND, it has been completely patched and updated, thus eliminating the vulnerability. However, it is also possible that the host was left an at un-patched, default installation.

Network Countermeasures = 3. There are obviously a few security mechanisms in place on this network, as evidenced by the fact that these packets were discovered and logged by the IDS. However, the border device appears to be allowing port 53 traffic to enter the network to too many different hosts (though it is possible that all 45 targeted IP addresses were DNS servers, it doesn't seem likely that all of them would need to be accessed by the outside world). The potential to lock down the network is there, but it appears that security needs to be tightened.

9. Defensive Recommendation:

Being that a fully patched and updated version of BIND is not usually vulnerable to common exploit attacks, the first recommendation would of course be to ensure that all servers running BIND are fully patched. As well, any machines that are running BIND as an unnecessary service should immediately have this service disabled (as well as any other unnecessary services).

A full audit of the border Cisco router/firewall should be performed to ensure that it is only allowing port 53 traffic to those servers which need to offer it to the outside world. (Since

this is a Cisco device, access lists can easily be applied to the external interface to restrict port traffic.) If there are indeed any servers that are running externally accessible DNS services, it is generally good practice to put these servers behind a firewall in a DMZ or screened subnet in order to better contain any potentially harmful traffic. This will allow the security administrator to further restrict port 53 traffic into the internal network. (It should be noted that DNS port 53 traffic commonly uses both tcp and udp packets, so both protocols must be accounted for.) This will also facilitate the organization's ability to restrict DNS information that is available to the outside world (aka "split-horizon" DNS), while not limiting internal access. For more information on split-horizon DNS, see http://homepages.tesco.net/~J.deBoynePollard/FGA/dns-split-horizon.html.

Whenever possible and practical, it is wise to remove and/or obfuscate the data which would be returned to a version.bind request; misleading and/or non-existent version information would make a potential hacker's job significantly more difficult in this case.

For more details on BIND specific security, see the BIND Security Matrix found at http://www.isc.org/index.pl?/sw/bind/.

In addition to the above mentioned recommendations, a system for regularly identifying and applying patches, service packs, and anti-virus updates should be designed and implemented on the network, in order to keep internal machines, routers, and switches as protected as possible and thus decrease their risk of initial exploitation. Regular network audits should also be performed in order to ensure continued security.

10. Multiple choice test question:

Based on the packet sample below, which of the following tools could have been used to create this packet?

```
10:43:10.364488 IP 217.131.191.70.4197 > 78.37.225.167.53:  4660
[b2&3=0x80] TXT CHAOS? version.bind. (30)
4500 003a 9b41 0000 2c11 7121 d983 bf46        E..:.A..,.q!...F
4e25 e1a7 1065 0035 0026 385e 1234 0080        N%...e.5.&8^.4..
0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0462 696e 6400 0010 0003                       .bind.....
```

A) Nessus
B) NMap
C) Nslookup
D) All of the above

Answer: D, All of the above.

Nessus: the "General: Determine which version of BIND name daemon is running" plugin would need to be enabled.

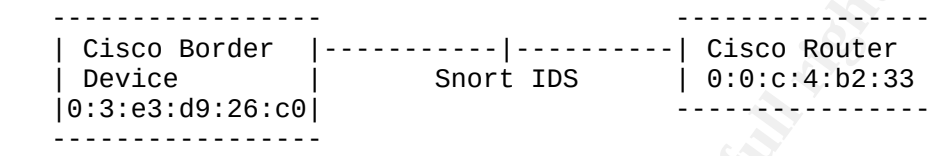
`nslookup –q=TXT –class=CHAOS version.bind. <target host>`

`nmap –sV <target host>`

SCAN nmap TCP, Practical Detect #3

1. Source of Trace:
   This capture file, 2002.6.10, was obtained from the raw logs at www.incidents.org/logs/raw and was posted July 19, 2002.

   Network Architecture:
   Though the exact network architecture is unknown, it can be surmised that the layout is similar to the following:



   This assumption is made based on the fact that all traffic originates from two MAC addresses, both of which are owned by Cisco Systems, according to http://www.coffer.com/mac_find/, and Ethereal version 0.9.9. A sample packet, captured with the following windump command, illustrates this:

```
windump -nr 2002.6.10 -e src port 80 and dst port 80
```

   -n = Do not resolve hostnames
   -r = Read from binary file
   -e = Display link layer information (MAC addresses)
   src port = Source Port
   dst port = Destination Port

```
16:24:56.534488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: IP
194.78.59.253.80 > 46.5.185.17.80: . ack 0 win 1400
```

   It can also be assumed that the Cisco device with the physical address of 0:3:e3:d9:26:c0 is a border router/firewall, since all traffic originating from this address is from public IP address space, and is destined for the internal address space of 46.5.0.0/16 (a standard Class B network structure).

2. Detect was generated by:
   Snort Intrusion Detection System version 2.1.1, running in full alert mode, with all rules included in snort.conf. Checksums were ignored in the file since the internal IP addresses and checksums were obfuscated before posting (www.incidents.org/logs/raw/readme). Snort was executed using the following command:

```
snort –A full –k none –c c:\snort\etc\snort.conf –l c:\snort\log
–r c:\snort\2002.6.10
```

   -A = Define alert mode (full, fast, console, or none
   -k none = Ignore checksums
   -c = Path to rule file

-l = Logging directory
-r = Read from binary file

A total of 120 alerts were logged relating to this detect, from 21 unique IP addresses. The following is an example of the alerts that were logged:

```
[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-16:25:01.534488 194.78.59.253:80 -> 46.5.185.17:80
TCP TTL:46 TOS:0x0 ID:56024 IpLen:20 DgmLen:40
***A**** Seq: 0x4A  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

Which was triggered by the following rule in the \snort\rules\scan.rules file (the EXTERNAL_NET and HOME_NET variables are both set to "any" in the snort.conf file):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP"; stateless; flags:A,12; ack:0; reference:arachnids,28;
classtype:attempted-recon; sid:628; rev:3;)
```

In reference to the sample packet above, this alert was triggered based on the following criteria:

A) It is a TCP packet with the ACK flag set (*A,12*—the 12 value tells the rule trigger on an ACK flag, regardless of the values of the reserved bits).
B) It has a TCP acknowledgement number (ack) value of 0. (Note: There are packets in this capture that trigger this alert, but seem to have an ack value of 1. In truth, this is windump using relative sequence numbers; when the –S [show absolute sequence numbers] switch is used, all ack values report as 0.)

3. Probability the source address was spoofed:
It has been noted by Whitehats Network Security Resource that, since this a reconnaissance attack, it is likely that the attacker will expect a reply, thus lowering the chances that the source addresses are spoofed.[30] However, there is no evidence that these packets are part of an established TCP session. Based on the data from SnortSnarf version 021111.1, it seems that the scans are actually divided in four groups of addresses (see chart below).

GROUP A: The first group of source addresses is characterized by a source and destination port of 80, and TTL values that are consistent with a *nix host.[31] When the packets from Group A are examined in more detail, it appears that all of the sequence numbers increment normally per source address. The TTL values are not static, indicating that the packets originated from different sources. It would appear that these addresses are the result of an automated tool of some kind, since they have a consistent acknowledgement number of 0, as well as a static window size (1400). Though the packets are crafted, I do not believe that the source addresses were spoofed, due to the

[30] Whitehats Network Security Resource, IDS28
[31] Zalewski, Michal

variances in TTL values between source addresses, as well as the incremental sequence numbers. The following is sample packet capture from Group A (checksum data has been truncated to save space).

```
17:34:55.944488 IP (tos 0x0, ttl 46, id 40735, len 40)
202.29.28.1.80 > 46.5.84.7.80: ack 0 win 1400
17:35:00.944488 IP (tos 0x0, ttl 46, id 40962, len 40)
202.29.28.1.80 > 46.5.84.7.80: ack 0 win 1400
17:35:07.274488 IP (tos 0x0, ttl 46, id 41294, len 40)
202.29.28.1.80 > 46.5.84.7.80: ack 0 win 1400
```

GROUP B: The second group of source addresses is characterized by a source port of 80 and a destination address and port of 46.5.80.149 port 6346. Of the six unique source addresses, each sent exactly two packets to the target host, all within a very close time frame (1-2 seconds). An ARIN lookup on the source addresses revealed that they all belong to different organizations; as well, TTL values do vary between different source hosts. TCP sequence numbers also increment per source address, though this is not a reliable test of validity, since there are only two packets present from each host. The packets are obviously related (same target host and port), though the windows sizes do vary, indicating that the packets are not crafted. I believe that these addresses are not spoofed, based on the evidence mentioned above. The following is a sample packet capture from Group B:

```
20:03:34.084488 IP (tos 0x0, ttl 47, id 21436, len 40)
64.3.83.34.80 > 46.5.80.149.6346: ack 0 win 1024
20:03:39.084488 IP (tos 0x0, ttl 47, id 21798, len 40)
64.3.83.34.80 > 46.5.80.149.6346: ack 0 win 1024
```

GROUP C: The third group of source addresses is characterized by a source port of 8* and a static destination port of 64236. As well, the TTL value is a constant 51 (with the exception of a single TTL of 50, which could be due to a normal route variance). An ARIN lookup on the source addresses revealed that they all belong to UUNET Technologies, though it seems unlikely that this many different source addresses could all be exactly the same distance away, relative to our home network. Only one packet was received from each source and all 12 of the packets captured in Group C were received within 1 second of each other. The sequence numbers of the packets increment across the entire group of packets—as if they all originated from a single physical machine, and the window size and ack values are also static. For these reasons, I believe that the packets in Group C are crafted and the addresses spoofed, though it is possible that at least one of the addresses is not, so that the attacker can still receive a reply (it is impossible to determine which, without a full traffic capture). The following is a sample packet capture from Group C:

```
10:00:47.304488 IP (tos 0x0, ttl 50, id 55586, len 40)
63.102.0.226.80 > 46.5.180.250.64236: ack 0 win 1400
10:00:47.384488 IP (tos 0x0, ttl 51, id 55598, len 40)
65.221.149.98.82 > 46.5.180.250.64236: ack 0 win 1400
10:00:47.384488 IP (tos 0x0, ttl 51, id 55597, len 40)
65.221.149.66.81 > 46.5.180.250.64236: ack 0 win 1400
```

GROUP D: The fourth group of packets contains only two source addresses and a total of 3 packets captured. The target host and port of all three packets is 46.5.180.150 port 53 (DNS). All three packets were received within 1 second of each other, and an ARIN lookup on the source addressed revealed that they are both registered to AllMusic.com (by Level 3 Communications). The sequence numbers from the first host increment as expected, and the TTL values are consistent between the two packets. The third packet (from the other source host) has the same TTL value as the previous two; because there is only one packet, analysis is difficult. I believe it is most likely that the first source address is genuine and the second is spoofed (with a randomly generated sequence number) and originated from the same host as the first two packets. The following is the packet capture of Group D:

```
16:56:37.544488 IP (tos 0x0, ttl 49, id 59820, len 40)
64.152.70.68.80 > 46.5.180.250.53: ack 0 win 1400
16:56:37.544488 IP (tos 0x0, ttl 49, id 59821, len 40)
64.152.70.68.53 > 46.5.180.250.53: ack 0 win 1400
16:56:37.644488 IP (tos 0x0, ttl 49, id 50414, len 40)
63.211.17.228.80 > 46.5.180.250.53: ack 0 win 1400
```

SnortSnarf Packet Analysis:

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)	SRC Port	DST Port	TTL
202.29.28.1	22	22	8	8	80	80	46
202.96.52.99	16	16	8	8	80	80	44
218.96.62.2	16	16	8	8	80	80	43
163.23.190.2	15	15	15	15	80	80	46
212.88.236.2	12	12	6	6	80	80	40
194.78.59.253	12	12	6	6	80	80	46
194.52.177.9	6	6	3	3	80	80	38
64.3.83.34	2	2	1	1	80	6346	47
209.6.58.139	2	2	1	1	80	6346	49
12.99.244.2	2	2	1	1	80	6346	46
65.113.31.2	2	2	1	1	80	6346	45
64.119.138.2	2	2	1	1	80	6346	46
206.111.234.194	2	2	1	1	80	6346	47
208.196.167.130	1	1	1	1	84	64236	51
65.221.149.98	1	1	1	1	82	64236	51
65.208.249.98	1	1	1	1	80	64236	51
65.221.167.34	1	1	1	1	83	64236	51
65.221.149.66	1	1	1	1	81	64236	51
63.102.0.226	1	1	1	1	80	64236	50
64.152.70.68	2	2	1	1	80, 53	53	49
63.211.17.228	1	1	1	1	80	53	49

Group A Group B Group C Group D

4. Description of Attack:

This attack is classified primarily as a reconnaissance attack, meaning that its main goal is not to actually perpetrate an attack, but rather to gather information to be used in a future attack. According to the NMap Man page, this attack is most often used to discover a firewall ruleset. [32] In order to do this, a TCP packet with the ACK flag set is sent to the target host; if the target host responds with a reset packet (as a properly configured host should, according to RFC 793), that port is considered "unfiltered" since the host received the packet and responded. [33] If no response is received from the target, it is assumed that there is a firewall and/or filtering device in front of the target, which dropped the packet. In this manner it can be determined which ports are available for attack on any given host; this attack cannot be used to map open ports (since any port that is unfiltered will respond with a RST), but it is a good way to find out if a host alive and potentially vulnerable.

5. Attack Mechanism:

I believe that this capture file actually contains a combination of genuine attacks and false positives. Once again, the "attack" patterns can be divided into four groups, which correspond with the four groups of source addresses mentioned above.

GROUP A: The first group of alerts is actually the most puzzling, in terms of deciphering the purpose of these packets. As mentioned above, it appears that the packets are not spoofed; it is unarguable that they are at least suspect, especially originating from port 80 (HTTP) to port 80—this type of traffic is rarely, if ever, seen in normal TCP communication. However, other than the source and destination ports, there is little else to relate the packets to one another. They originate from all over Europe and Asia, and usually send two packets to each targeted host, those this can vary (refer to SnortSnarf chart above). I did find evidence that these packets may be part of a load balancing system, based on a post by Chris Brenton. [34] He stated that he saw similar traffic patterns from a Radware device performing load balancing; however, there were other packets to other ports associated with the traffic, and all of the traffic originated from a single source. Because of this conflict in evidence, I believe that it is more likely that these packets are part of a distributed scan being performed by a single attacker that "owns" a wide range of hosts. The following is a sample of the alerts captured:

```
[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-16:25:06.534488 212.88.236.2:80 -> 46.5.185.17:80
TCP TTL:40 TOS:0x0 ID:56288 IpLen:20 DgmLen:40
***A**** Seq: 0xAE  Ack: 0x0  Win: 0x578  TcpLen: 20

[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-16:25:11.534488 212.88.236.2:80 -> 46.5.185.17:80
TCP TTL:40 TOS:0x0 ID:56548 IpLen:20 DgmLen:40
***A**** Seq: 0x110  Ack: 0x0  Win: 0x578  TcpLen: 20
```

[32] Insecure.com LLC
[33] Information of Sciences Institute
[34] Brenton, Chris

GROUP B: The second group of alerts appears to actually be a false positive, though they still may be of concern (more of a political motivation than security). Since all packets in this group are directed to the same host at port 6346, I did some research and found out that Gnutella, a popular music download client, listens on this port (and several others). [35]I believe that the packets captured here are the result of several Gnutella servers and/or clients performing a check to see if this host is alive. I could not find any information specifically on Gnutella and how it the target host would react if this ACK packet was received, but it seems apparent that this is not necessarily an attack (though it may be against corporate policy to allow controversial programs such as this on company computers), but rather a "Hello? Are you still out there and using our product?" A previous detect by Johnny Wong, with a full packet capture detailing the Gnutella connect sequence supports this theory. [36] The following is a sample of the alerts captured:

```
[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-20:03:34.084488 64.3.83.34:80 -> 46.5.80.149:6346
TCP TTL:47 TOS:0x0 ID:21436 IpLen:20 DgmLen:40
***A**** Seq: 0x191  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-20:03:39.084488 64.3.83.34:80 -> 46.5.80.149:6346
TCP TTL:47 TOS:0x0 ID:21798 IpLen:20 DgmLen:40
***A**** Seq: 0x1CF  Ack: 0x0  Win: 0x400  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

GROUP C: The third group of alerts does appear to be an actual reconnaissance attack, though the attacker was very specific in what they were looking for: port 64236. I was unable to find any common services, backdoors, or worms that are associated with this port, so I can only assume that the attacker was looking for a specific vulnerability or process (possibly from his own rootkit). As mentioned above, this type of scan only tells the attacker that the host is alive, and we are unable to see whether or not any of the hosts responded to his query, since we do not have a full traffic capture. It think it is safe to assume that the attacker was already aware of the fact that this firewall/border router was not filtering port 64236, and was looking for a previously compromised host (especially since only packet was sent per source host). The following is a sample of the alerts captured:

```
[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/10-10:00:47.384488 208.196.167.130:84 -> 46.5.180.250:64236
TCP TTL:51 TOS:0x0 ID:55600 IpLen:20 DgmLen:40
***A**** Seq: 0x2E1  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

[35] Zalewski, Michal
[36] Wong, Johnny

```
[**] [1:628:3] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/10-10:00:47.384488 65.208.249.98:80 -> 46.5.180.250:64236
TCP TTL:51 TOS:0x0 ID:55596 IpLen:20 DgmLen:40
***A**** Seq: 0x2DD  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

GROUP D: The last group of alerts also appears to be a genuine reconnaissance attack, most likely generated by NMap or a comparable tool. The attacker appears to be looking for a host that is unfiltered on port 53, the common port for DNS. There were not any further packets captured from the two offending hosts to indicate whether this reconnaissance was followed by an actual attack, or even further reconnaissance (there were many version.bind requests in this packet capture, but none directed at this host). Upon further analysis, it appears that this may also have been an attempt to see if a previously compromised host was alive and listening. Roughly 1 hour after the first three reconnaissance packets were received, this host suddenly starting spewing out suspicious port 80 packets to various hosts. Exactly 370 alerts were recorded on these packets, most of which were BARE BYTE UNICODE alerts, although other alerts were also captured, including OVERSIZE REQUEST-URI DIRECTORY, DOUBLE DECODING, and SHELLCODE attacks. It would appear that this host was already compromised at the date of this packet capture, and that the scan packets were some type of initiation and/or just a connectivity test of the compromised 46.5.180.250 host. (The exact details of this compromise are out of the scope of this detect, as they are a detect in themselves—it is noted that if this were a "live" packet capture, immediate further research would be highly recommended, but in the interest of space, it has been omitted here.)

As for the exact tool used to create these attacks, I suspect that an older version of NMap was used (I say older version because newer versions of NMap generate "seemingly random" sequence and acknowledgement numbers when performing scans such as these—for this reason I was unable to exactly duplicate the packets in this capture using NMap).[37] Tools such as Nessus are also capable of performing this type of scan, since Nessus contains an NMap plugin.

6. Correlations:

More information on the SCAN nmap TCP signature and alert can be found in the arachNIDS database at Whitehats Network Security Resource (http://www.whitehats.com/info/IDS28).

The CVE number for this attack is CAN-1999-0523; the entry is currently under review, mostly because the reviewers have come to the conclusion that the description "ICMP echo (ping) is allowed from arbitrary hosts" is too general. [38]

The user list discussion with Chris Brenton regarding load balancing can be found at http://www.incidents.org/archives/intrusions/msg08129.html.

[37] Insecure.com LLC
[38] Common Vulnerabilities and Exposures

More information on NMap can be found on the Nmap Man Page (http://www.insecure.org/nmap/data/nmap_manpage.html).

Since this a public detect posted to incidents.org, many other GCIA students have submitted entries relating to similar alerts. The following entries and their replies were examined while researching this subject:
Cori Lynn Arnold:
http://cert.uni-stuttgart.de/archive/intrusions/2004/03/msg00071.html

Johnny Wong:
http://www.giac.org/practical/GCIA/Johnny_Wong_GCIA.pdf

7. Evidence of Active Targeting:
Though all of the target hosts in this packet are members of the 46.5.0.0/16 internal network, it does not appear that any specific subnet boundaries or incremental values were used in the scans. Group B's Gnutella traffic was certainly targeted at the 46.5.80.149 address, which is apparently a known Gnutella host. It is also possible that the hosts targeted by Groups C and D were specifically chosen, possibly because the attacker already knew of a vulnerability or compromise (this would seem especially likely of the 46.5.180.250, which was the target of Group D). Without a full traffic capture to determine how the hosts responded, this cannot be fully determined.

8. Severity:
* Severity = 1 *

Severity = (Criticality [4] + Lethality [2]) –
(System Countermeasures [2] + Network Countermeasures [3])

Criticality = 4. All told, this scan traffic targeted 42 unique hosts on our internal network, of which the services and function are unknown and difficult to determine without a full packet capture. Some of the targeted hosts were most likely workstations, though servers could have also been targeted. For this reason, criticality has been rated as a 4, since it is possible that web, DNS, domain controllers, or file servers could have potentially been targeted.

Lethality = 2. The reconnaissance scans detected here do not give any specific indication of the intent of the attacker, once the data has been gathered. The attacker may be looking for a previously compromised host, or just checking to see if a host is "alive." It does appear that at least one of the hosts may have been previously compromised (host 46.5.180.250), but the severity of the compromise is also unknown. For these reasons, I've given Lethality a 2, assuming that some damage could potentially be done to a compromised target host.

System Countermeasures = 2. Without a full traffic capture, this is difficult to determine. If the systems have been properly hardened, updated, and patched a compromise is less likely. However, it is also possible that the targeted systems are running an unpatched,

default installation, or any number of vulnerable services. Since it appears that at least one system has already been compromised, I have assumed the setup is closer to the latter.

Network Countermeasures = 3. It is apparent that there are a few security mechanisms in place on this network, as evidenced by the fact that these packets were discovered and logged by the IDS. However, the border device appears to be allowing unnecessary port traffic (such as 6346 and 64236) to enter the network. Based on various other alerts included in this capture (including version.bind requests), it would seem that the device has not been properly configured to protect the internal network.

9. Defensive Recommendation:

Most, if not all, of the malicious traffic captured in this file could have been prevented from reaching the target hosts by a properly configured border device. It is unlikely that inbound port 6346 and port 64236 traffic is necessary to the functionality of this network. If one has not already been installed, a firewall should be installed and configured on the border of this network, and filtering rules should be implemented that will only allow that traffic which it is absolutely necessary to permit from the external network (such as DNS, SMTP, HTTP, etc.). Since this appears to be a Cisco shop, this filtering could be easily implemented by using access lists on the external interface. It is also a good idea to configure the firewall to drop "suspicious" traffic, such as packets with a source and destination port 80, and traffic originating from reserved addresses (192.168.x.x, 10.x.x.x., or broadcast addresses).

Also, if not already configured, client permissions on the domain workstations should be implemented in such a way that the user is unable to install unsupported software (such as Gnutella), which will help eliminate unnecessary traffic and bandwidth usage on the network, as well as protect against any security exploits that may be associated with these programs. If client permissions cannot be changed, a system should be implemented for auditing the workstations on a regular basis, to check for any unexpected file changes and/or program installations.

In addition to the above mentioned recommendations, a process for regularly identifying and applying patches, service packs, and anti-virus updates should be designed and implemented on the network, in order to keep internal machines, routers, and switches as protected as possible and thus decrease their risk of initial exploitation. Regular network audits should also be performed in order to ensure continued security.

10. Multiple choice test question:

Based on the packet sample below, which of the following characteristics are the most suspicious, and are possibly evidence of a crafted packet?

```
12:09:58.204488 IP (tos 0x0, ttl 41, id 1068, len 40)
212.88.236.2.80 > 46.5.249.63.80: . ack 0 win 1400
4500 0028 042c 0000 2906 ab0c d458 ec02        E..(.,..)....X..
2e05 f93f 0050 0050 0000 023e 0000 0000        ...?.P.P...>....
5010 0578 c4e6 0000 0000 0000 0000             P..x.........
```

E) A source and destination port of 80
F) A TTL value of 41
G) An acknowledgement number of 0
H) Both A and C

Answer: D, Both A and C

Though it is possible to have a source and destination port of 80, it is not usually seen in "normal" TCP traffic. The same applies to a TCP acknowledgement number of 0. [39]

[39] Brenton, Chris--SANS Track 2

References: Practical Detects

Binde, Beth. "GIAC Intrusion Detection In-Depth." GIAC GCIA Practical Assignment version 3.3. 12 May 2003. URL: http://www.giac.org/practical/GCIA/Beth_Binde_GCIA.pdf.

Brenton, Chris. "Re: Anyone else seeing TCP ACKs on port 80? –Not LION worm." Online posting. Incidents.org. 26 April 2002.
URL: http://www.incidents.org/archives/intrusions/msg08129.html.

Brenton, Chris. "SANS Track 2: Firewalls and Perimeter Protection." San Diego, CA. January 2004.

CERT/CC. "Multiple Vulnerabilities in lpd." CERT Coordination Center. CA-2001-30. 15 November 2001. URL: http://www.cert.org/advisories/CA-2001-30.html.

Common Vulnerabilities and Exposures. "CAN-1999-0009." CAN-1999-0009. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0009.

Common Vulnerabilities and Exposures. "CAN-1999-0523 (under review)." CAN-1999-0523. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0523.

Common Vulnerabilities and Exposures. "CAN-1999-0660 (under review)." CAN-1999-0660. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0660.

Common Vulnerabilities and Exposures. "Search Results: BIND." 24 May 2004. URL: http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=BIND.

Gordon, Les. "What is the Q Trojan?" SANS Intrusion Detection FAQ.
URL: http://www.sans.org/resources/idfaq/qtrojan.php.

Information of Sciences Institute. "RFC 793—Transmission Control Protocol." RFC-Editor Webpage. September 1981. URL: ftp://ftp.rfc-editor.org/in-notes/rfc793.txt.

Insecure.com LLC. "Nmap security scanner man page."
URL: http://www.insecure.org/nmap/data/nmap_manpage.html.

Mogul, Jeffrey. "RFC 919—Broadcasting Internet Datagrams." RFC-Editor Webpage. October 1984. URL: ftp://ftp.rfc-editor.org/in-notes/rfc919.txt.

"Services file." Available on Track 2: Firewall, Perimeter Protection & Virtual Private Networks by Chris Brenton (CD-ROM). Version 2.1. 14 November 2003.

The Snort Project. "Snort Users Manual." Version 2.1.1.
URL: http://www.snort.org/docs/snort_manual/.

Whitehats Network Security Resource. "IDS28—'PROBE-NMAP_TCP_PING." arachNIDS Intrusion Detection Database.

Wong, Johnny. "GIAC Intrusion Detection In-Depth." GIAC Practical Assignment Version 3.3. Document version 2.0. URL: http://www.giac.org/practical/GCIA/Johnny_Wong_GCIA.pdf.

Zalewski, Michal. "pof—passive OS fingerprinting." Available on Track 2: Firewall, Perimeter Protection & Virtual Private Networks by Chris Brenton (CD-ROM). Version 2.1. 14 November 2003.

Analyze This

Executive Summary:

The following security log audit was performed for the University of Maryland, Baltimore County. It is the conclusion of the auditor that the security measures provided by the University's staff are very satisfactory, though there is always room for improvement in the ever-changing world of network security.

Much of the log data generated in these files is the result of broad IDS rules that are capturing unnecessary data—suggestions for improving the IDS signatures are included for the most prevalent alert entries. In addition, some of the log data provided here is indicative of hostile attackers and/or scanners or viruses, some of which have apparently succeeded in compromising hosts on the network. In these cases, immediate action should be taken to sanitize and patch the compromised machines.

The following text describes the auditor's suggestions for improving the Snort signature database, as well as additional defensive recommendations.

Files Analyzed:

The following files were downloaded from www.incidents.org/logs and used in this analysis. They range in date from February 18, 2004 through February 22, 2004.

Alerts	Scans	OOS
alert.040218.gz	scans.040218.gz	oos_report_040218.txt
alert.040219.gz	scans.040219.gz	oos_report_040219.txt
alert.040220.gz	scans.040220.gz	oos_report_040220.txt
alert.040221.gz	scans.040221.gz	oos_report_040221.txt
alert.040222.gz	scans.040222.gz	oos_report_040222.txt

Initial analysis of the log files above indicates that the internal network space (referred to in the alerts files as MY.NET) is actually in the 130.85.0.0/16 address space, which is registered to the University of Maryland Baltimore County (130.85.0.0 addresses were present in the scans log, which apparently had not been obfuscated). UMBC has several DNS servers registered (umbc3.umbc.edu – umbc6.umbc.edu), which are in the IP address range of 130.85.1.3-1.5 and 130.85.6.7. The main UMBC webpage is registered at the 130.85.24.34 address.

Alert Detection Analysis:

A total of ten alerts from the alert logs were analyzed in greater detail; these alerts were chosen based on the fact they were the top ten most prevalent entries in the alert log files. The files were processed using SnortSnarf, and the following signatures were noted to have acquired the most traffic during this five day period:

Alert Signature	# Alerts	# Sources	# Destinations
SUNRPC highport access!	34833	27	14312
TCP SRC and DST outside network	19309	17048	44
130.85.30.4 activity	11305	299	1
Possible trojan server activity	7786	42	238
SMB Name Wildcard	5567	175	483
Incomplete Packet Fragments Discarded	4985	80	56
130.85.30.3 activity	4586	173	1
High port 65535 tcp - possible Red Worm - traffic	3663	82	103
EXPLOIT x86 NOOP	1659	187	91
High port 65535 udp - possible Red Worm - traffic	599	87	64

Alert #1 – SUNRPC highport access!
Alerts: 34,833
Source Machines: 27
Machines Targeted: 14,312

Top 5 Offenders: SUNRPC highport access!				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
210.98.224.82	34653	34662	14309	14311
166.90.150.29	64	64	1	1
203.15.51.51	17	17	1	1
213.87.4.1	15	15	4	4
216.239.41.99	13	13	3	3

Severity: Medium. This type of reconnaissance scan may be looking for hosts that are either open to compromise or have been previously compromised by exploiting RPC vulnerabilities. There are a number of different vulnerabilities for the RPC (Remote Procedure Call) services, ranging from service disruption to gaining root level access to the box.[40] However, there is no data in the files collected here (in the form of responses from targeted hosts) to indicate that any computers have been compromised.

False Positives: It is possible that an application will legitimately communicate at this port and could generate a false positive.

Description: Though there are a considerable number of Snort signatures based on RPC access, this alert appears to have been customized by the University Administrators, as the signature

[40] Whitehats Network Security Resource, Search for RPC 32771

output is not standard. It would appear that this rule has been modified to read something similar to:

alert tcp $EXTERNAL_NET any -> $HOME_NET 32771 (msg: “SUNRPC highport alert!”;)

The number one offender for this alert was the source address 210.98.224.82, which also triggered several other alerts (including External FTP to HelpDesk and 30.3 and 30.4 access). A name lookup on the address has it registered to Booil Mobile Telecom Corporation (Korea), and DShield reports the DNS name as bint.eyes.co.kr, neither of which give much insight into the origin of the attack. This attacker scanned over 14,000 addresses on the University subnet for port 32771 (a known RPC port, especially on Solaris boxes). It doesn't appear that any of the addresses scanned responded directly to this host (two addresses, 130.85.150.44 and 130.85.150.198 did send SMB Name Wildcard alerts to this host, though not in response to the RPC scan). It is recommended that this address be further monitored for malicious traffic (or possibly even blocked at the perimeter, if policy permits), and the Telecom abuse representative contacted.

The remaining four of the Top 5 offenders are a great a concern, though they should still be monitored for further activity; it should be noted that the addresses appear to be spoofed (it seems unlikely that Google and the Spam and Open Relays Blocking System are attacking the network, though it is technically possible—the auditor is unaware of any legitimate traffic that is generated by these sources at this port).

Alert #2 – TCP SRC and DST outside network
Alerts: 19,309
Source Machines: 17,048
Machines Targeted: 44

Top 5 Offenders: TCP SRC and DST outside network				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
172.160.153.224	18	18	5	5
192.168.1.100	9	9	4	4
172.146.55.229	8	8	1	1
192.168.1.22	7	7	3	3
68.33.227.200	6	6	3	3

Severity: Low. The presence of a source and destination address in a packet that does not originate from the internal network subnet (yet is still captured inside the network) can indicate the use of a packet crafting tool to spoof addresses, or the use of reserved addresses such as 192.168.0.0 or 169.254.0.0 which occur in normal Windows traffic.

False Positives: By definition, this rule cannot have any false positives. However, it does create a lot of unnecessary log data.

Description: This also appears to be a custom rule, created by the University. It can be surmised that the rule is similar to the following:

alert tcp !HOME_NET any -> !HOME_NET any (msg: "TCP SRC and DST outside network";)

The number one offender for this alert is source address 172.160.153.224, which is registered to AOL. Traffic from this address was restricted to this signature only, and it appears that it targets various IP addresses at port 80. Whether this is a crafted packet with malicious intent, or just a misconfigured NIC (perhaps from someone's home PC) trying to register software, is unclear. Due to the low occurrence and low risk to the network, this is not high priority traffic. Such is the case for the rest of the Top 5 as well.

Most of the alert traffic for this signature actually comes from a multitude (around 17,000 to be exact) of 169.254.0.0 addresses, with single packet traffic. The 169.254.0.0/16 subnet is reserved by IANA, and is used by Windows machines as a default IP subnet when a DHCP server is unavailable. [41] Machines with multiple Ethernet cards (where only one card is actually connected to the network) could generate this type of traffic. It is also noted that a normal Windows host trying to participate in SMB file sharing will generate traffic, not only from its own valid IP address, but also from a 169.254.x.x or 192.168.1.x address as well (which helps account for the unusually large number of SMB Name Wildcard alerts that are also seen in these logs from 169.254.x.x addresses—see Alert #5). [42]

The University security administrators should consider tuning this rule so that it does not include this type of normal traffic. This could be done using a variable such as $HOME_NET_NOISE, which would be defined as a range of IPs. For example:

var HOME_NET_NOISE 130.85.0.0/16,169.254.0.0/16,192.168.1.0/24

alert tcp !HOME_NET_NOISE any -> !HOME_NET_NOISE any (msg: "TCP traffic outside network, not noise";)

[41] Microsoft Corporation.
[42] Martin, Daniel

Alert #3 – 130.85.30.4 activity
Alerts: 11,305
Source Machines: 299
Machines Targeted: 1

Top 5 Offenders: 130.85.30.4 Activity				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
68.50.102.64	3378	3378	1	1
68.55.62.244	1000	1000	1	1
12.21.173.176	958	1712	1	2
216.83.163.132	667	667	1	1
68.55.179.193	521	521	1	1

Severity: Not Applicable. This alert seems to target any and all activity to the 130.85.30.4 host, which is registered as lan2.umbc.edu in DShield. Severity is "Not Applicable" since specific attack traffic is not defined.

False Positives: This rule, by definition, has no false positives. However, it does create a lot of unnecessary log data.

Description: The 130.85.30.4 Activity rule is a custom rule created by the University. Since no internal IP addresses triggered this alert, it is assumed to be similar to the following:

alert tcp $EXTERNAL_NET any -> 130.85.30.4/32 any (msg: "130.85.30.4 Activity";)

When this address was input into DShield, its DNS name was reported as lan2.umbc.edu. When visiting this address in a web browser, a "Novell NetStorage" login screen was reported, as well as a description of how to access user network drives when out of the office. It seems safe to assume that this site provides network share access via the web to users when they are away from the office. It also seems obvious that these users, when they are out of the office, will not be on the 130.85.0.0/16 LAN, and will therefore trigger the alert (this seems to be the case, since most of the addresses above are registered to Comcast Cable). This rule, by nature, generates a lot of unnecessary log data and should be disabled (if this is intended to be a public access server), with the assumption that a NIDS sensor is watching for other truly nefarious traffic targeted at this IP. If this server is restricted to only a certain set of users, or is not currently in production, the University should instead consider limiting access to the server at the firewall or border device, if policy permits. If more detailed intrusion detection is required for this host, the University should consider installing a Host-Based IDS on this server, with its own customized ruleset.

Alert #4 – Possible Trojan server activity
Alerts: 7,786
Source Machines: 42
Machines Targeted: 238

Top 5 Offenders: Possible Trojan server activity				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
130.85.29.3	3655	3658	1	4
169.200.215.36	3486	3486	1	1
24.86.3.160	135	135	133	133
24.128.171.138	125	125	125	125
130.85.111.197	108	118	1	2

Severity: Medium. This rule is configured to alert on any activity to TCP port 2734, a well known port that is commonly used by many Trojans, such as BadBlood, EGO, LION, and SubSeven, just to name a few. [43] If this traffic is genuinely related to one of these worms, the potential damage can be severe, including network downtime, low bandwidth availability, and cleanup and removal hours and resources.

False Positives: It is possible for this rule to generate false positives, as normal network traffic can occur at this port.

Description: This appears to be another custom rule written by UMBC. Based on the alert traffic analyzed, the rule is assumed to be similar to the following:

alert tcp any any -> any 27374 (msg: "Possible Trojan server activity";)

Judging by the sheer number of alerts reported from this rule, it would seem that the University is running rampant with Trojans! However, that is not the case. The number one offender, 130.85.29.3, is actually an application server (bb-app4.umbc.edu), which brings up a BlackBoard login page when visited (BlackBoard appears to be an interface for students to check and change their schedule, view their transcript, etc.). The server seems to be holding a conversation with a single host, 169.200.215.36 (offender #2) from http port 80 on the server to port 27374 on the client (see sample alert capture below).

[43] SnortSnarf

02/19-12:59:43.595361 [**] Possible trojan server activity [**] 169.200.215.36:27374 -> 130.85.29.3:80
02/19-12:59:43.595373 [**] Possible trojan server activity [**] 130.85.29.3:80 -> 169.200.215.36:27374
02/19-12:59:43.931810 [**] Possible trojan server activity [**] 169.200.215.36:27374 -> 130.85.29.3:80
02/19-12:59:43.931891 [**] Possible trojan server activity [**] 130.85.29.3:80 -> 169.200.215.36:27374
02/19-12:59:43.931821 [**] Possible trojan server activity [**] 169.200.215.36:27374 -> 130.85.29.3:80
02/19-12:59:43.969265 [**] Possible trojan server activity [**] 130.85.29.3:80 -> 169.200.215.36:27374

The traffic seems to be legitimate traffic from a web server to a client which happened to use port 27374 to establish its connection. The 169.200.215.36 address itself belongs to First Union National Bank Corporation and does not seem to be spoofed, since the length of the TCP conversation implies that the three-way handshake has taken place. The recommendation here is to scan the machine for good measure, though it is unlikely that it has been infected.

The third offender, 24.86.3.160, is considerably more suspicious, as it seems to be searching for machine on the network that has previously been compromised by a Trojan listening on this port. The address is registered to Shaw Communications (a telecommunications provider), and appears to have scanned 133 addresses on this subnet. Of note are the 3 addresses who replied to this query: 130.85.6.15 (remedy.umbc.edu), 130.85.190.203 (wt-vpn2.umbc.edu), and 130.85.190.202 (wt-vpn1.umbc.edu). These hosts should be scanned immediately for possible infection and closely monitored.

Alert #5 – SMB Name Wildcard
Alerts: 5,567
Source Machines: 175
Machines Targeted: 483

Top 5 Offenders: SMB Name Wildcard				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
130.85.11.7	3139	3140	1	1
130.85.75.13	526	527	167	168
130.85.150.198	325	325	147	147
130.85.150.44	214	214	93	93
130.85.11.6	195	195	2	2

Severity: Low. This traffic is known to be generated by Windows hosts during normal conversation. [44] The fact that all of this traffic originates internally is a good indication that this traffic is probably benign noise.

False Positives: False positives can occur during normal SMB communications, especially between windows hosts when attempting to enumerate available named pipes. [45]

Description: Since this is an older Snort system, that may explain the difference in the alert "message," but it is assumed that this is a "pre-packaged" Snort alert that is now known as "NetBIOS Name Query." [43] If this is the case, the alert is similar to the following:

alert UDP any any -> any 137 (msg: "SMB Name Wildcard"; content: "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00 00|"; classtype: info-attempt; reference: arachnids,177;)

When closely examined, it becomes immediately apparent that literally all of this traffic originated internally. This supports the theory that this traffic is simply noise that is safely ignored; however, it should be noted that this type of traffic can also be used to point out misconfigurations and/or malfunctioning machines. For example, hosts 130.85.11.6 and 11.7 seem to be generating an unusually large amount of this traffic to the 169.254.0.0 subnet—it appears that they may have misconfigured with incorrect name server addresses or domain information, and are searching for a suitable name space to belong to. The configurations on these machines should be verified, and the University should consider modifying this rule to only alert on inbound traffic, not internal traffic. For example:

alert udp $EXTERNAL_NET any -> $HOME_NET 137

Alert #6 – Incomplete Packet Fragments Discarded

Alerts: 4,985
Source Machines: 80
Machines Targeted: 56

Top 5 Offenders: Incomplete Packet Fragments Discarded				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
130.85.21.67	1782	1782	8	8
130.85.21.69	1566	1568	8	8
130.85.21.68	1482	1482	8	8
211.47.68.231	33	33	1	1
69.3.1.20	8	15	1	1

Severity: Low. This alert tends to be a false positive triggered on older Snort systems.

[44] Martin, Daniel
[45] Whitehats Network Security Resource, IDS177

False Positive: This type of alert can be generated by older Snort systems using the defrag preprocessor.

Description: This type of traffic is generated by the defrag preprocessor when packet fragments without an established TCP stream are discovered (in other words, packet fragments arrived that could not be reassembled into a whole). [46] It has been stated by Marty Roesch (one of the authors of the defrag preprocessor) that this version of the defrag preprocessor had some "fairly nasty failure modes."[47] This traffic is likely just noise, created by bandwidth lags, or inefficient applications. For example, the internal hosts that triggered this alert (130.85.21.67-69) all seem to be similar in function (since they are registered as c00149-c00150.umbc.edu, respectively), and all are all generating these packets. Before raising any red flags, it is recommended that the University upgrade Snort to the latest "frag2" preprocessor version.

Alert #7 – 130.85.30.3 activity
Alerts: 4,586
Source Machines: 173
Machines Targeted: 1

Top 5 Offenders: 130.85.30.3 activity				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
68.55.178.168	1370	1468	1	2
12.21.173.176	754	1712	1	2
131.92.177.18	504	504	1	1
68.55.27.157	385	550	1	2
68.55.148.5	299	322	1	2

Severity: Not Applicable. This alert seems to target any and all activity to the 130.85.30.3 host, which is registered as lan1.umbc.edu in DShied. Severity is "Not Applicable" since specific attack traffic is not defined.

False Positives: This rule, by definition, has no false positives. However, it does create a lot of unnecessary log data.

Description: The 130.85.30.3 activity rule is a custom rule created by the University. Since no internal IP addresses triggered this alert, it is assumed to be similar to the following:

alert tcp $EXTERNAL_NET any -> 130.85.30.3/32 any (msg: "130.85.30.3 Activity";)

When this address was referenced in DShield, its DNS name was reported as lan1.umbc.edu. When visiting this address in a web browser, a "Welcome to Netware 6" login screen was displayed, as well as options for Remote Manage, NetStorage, and iManager. It seems safe to assume that this site is a default administration site provided with Novell for remote access to

[46] LURHQ Threat Intelligence Group
[47] Roesch, Martin

network resources. It also seems obvious that the most common use for this site is probably going to be for IT personnel when doing remote administration, and that these users, when they are out of the office, will not be on the 130.85.0.0/16 LAN and will therefore trigger the alert (this seems to be the case, since most of the addresses above are registered to Comcast Cable). This rule, by nature, generates a lot of unnecessary log data and should be disabled (if this is indeed intended to be a public access server), with the assumption that a NIDS sensor is watching for other truly nefarious traffic targeted at this IP. Similarly to the 130.85.30.4 machine, if this server is restricted to only a certain set of users, or is not currently in production, the University should instead consider limiting access to the server at the firewall or border device, as policy permits. If more detailed intrusion detection is required for this host, the University should consider installing a Host-Based IDS on this server, with its own customized ruleset.

Alert #8 / 10 – High port 65535 tcp/udp – possible Red Worm - traffic
Alerts: 3,663 / 599
Source Machines: 82 / 87
Machines Targeted: 103 / 64

Top 5 Offenders: tcp - Possible Red Worm traffic				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
130.85.153.37	1317	1317	1	1
68.6.96.171	643	643	1	1
130.85.97.76	592	592	1	1
82.64.5.79	579	579	1	1
130.85.24.44	69	69	3	3

Top 5 Offenders: udp - Possible Red Worm traffic				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
130.85.42.8	53	53	7	7
130.85.84.164	46	46	7	7
61.25.24.181	42	42	3	3
61.22.170.200	42	42	1	1
220.42.72.103	36	36	3	3

Severity: Medium. The Adore (Red) Worm, which targets Linux machines, communicates on the high port of 65535. If infected, a system will scan for vulnerable hosts on random Class B subnets and then attempt to download the main program files from China. [48]

False Positives: It is possible that a host will communicate on this port during normal traffic exchanges.

[48] F-Secure

Description: The TCP and UDP forms of these alerts have been analyzed together, since they are similar in nature. These rules appear to have been customized by the University staff. Judging by the alert captures, they are assumed to read:

alert tcp any any -> any 65535 (msg: "Highport 65535 tcp – possible Red Worm – traffic)
alert udp any any -> any 65535 (msg: "Highport 65535 udp – possible Red Worm – traffic)

The Linux/Adore worm (otherwise known as the Red worm), will attempt to modify various files on an infected machine; it will also attempt to send sensitive information via email. The backdoor that is included with the worm does listen on port 65535 and is activated by a ping packet of a certain size. [48] The three internal hosts that are included in the TCP Top 5 Offenders list were the first to be analyzed. It appears that they are all public access servers of some type, being that they are registered as refweb08.umbc.edu, ppp-076.dialup.umbc.edu, and userpages.umbc.edu. Oddly enough, the first four of the five offenders are related—each External offender is having a conversation with the Internal offender which precedes it in the list (i.e. 130.85.153.37 is talking to 68.6.96.171). Since refweb08.umbc.edu and userpages.umbc.edu are both web servers and communicating to the offending targets on port 80, it would seem that this traffic is actually a benign false positive, since this is not a characteristic of the Adore worm. The same applies to ppp-076.dialup.umbc.edu, which appears to be communicating on port 3694, which is a common VPN communications port. [49]

As for the UDP traffic noted here, the top "offending" hosts that are causing these alerts mostly seem to be participating in various file sharing sessions; specifically WinMX file sharing on port 6257. Destinations do vary, but it would appear that this traffic does not carry the signatures of the Adore worm and is also benign.

It seems that the traffic generated from these rules are mostly false positives—it is recommended that the rules be tuned to included specific packet payload (such as the files downloaded or the web address in China), in order to decrease the number of false positives generated by this rule.

[49] SnortSnarf

Alert #9 – EXPLOIT x86 NOOP
Alerts: 1,659
Source Machines: 187
Machines Targeted: 91

Top 5 Offenders: EXPLOIT x86 NOOP				
Host	#Alerts (sig)	#Alerts (Tot.)	#Dest (sig)	#Dest (Tot.)
195.154.199.210	448	448	22	22
213.93.153.31	358	358	1	1
212.87.86.80	175	203	24	27
83.28.6.178	139	139	14	14
131.118.254.130	120	120	1	1

Severity: Medium. If a machine is susceptible to a NOOP buffer overflow vulnerability, it could be compromised in a various ways, including arbitrary code execution.

False Positives: According to Whitehats Network Security Resource, it is possible that the 0x90 sequence may occur in normal traffic. [50]

Description: This appears to be a standard snort rule (if older)--the "message" is now slightly different, but the alert is most likely the same:

alert UDP $EXTERNAL any -> $INTERNAL any (msg: "EXPLOIT x86 SHELLCODE NOOP"; content: "|90|"; classtype: system-attempt; reference: arachnids,362;)

At first glance this traffic seems to be a false positive, triggered by legitimate communication. Offender #2, 213.93.153.31, targets only one host, as does Offender #5. It is likely that this alert traffic is a false positive, as the conversations between the hosts and their targets are lengthy, and are on common ports (port 1495—CVC and port 119—NNTP). However, three of the top 5 offenders are considerably more suspicious. Offender #3 (212.87.86.80) is targeting many internal hosts at port 389, the well know port for LDAP, but also the well known port for BlackIce Defender. [51] This is considered more suspicious due to the Witty Worm (which, oddly enough, is also posted in a warning bulletin on UMBC's own website), and targets a vulnerability in BlackIce Defender and RealSecure. [52] It is to be assumed that this host is scanning for systems that are vulnerable to this exploit.

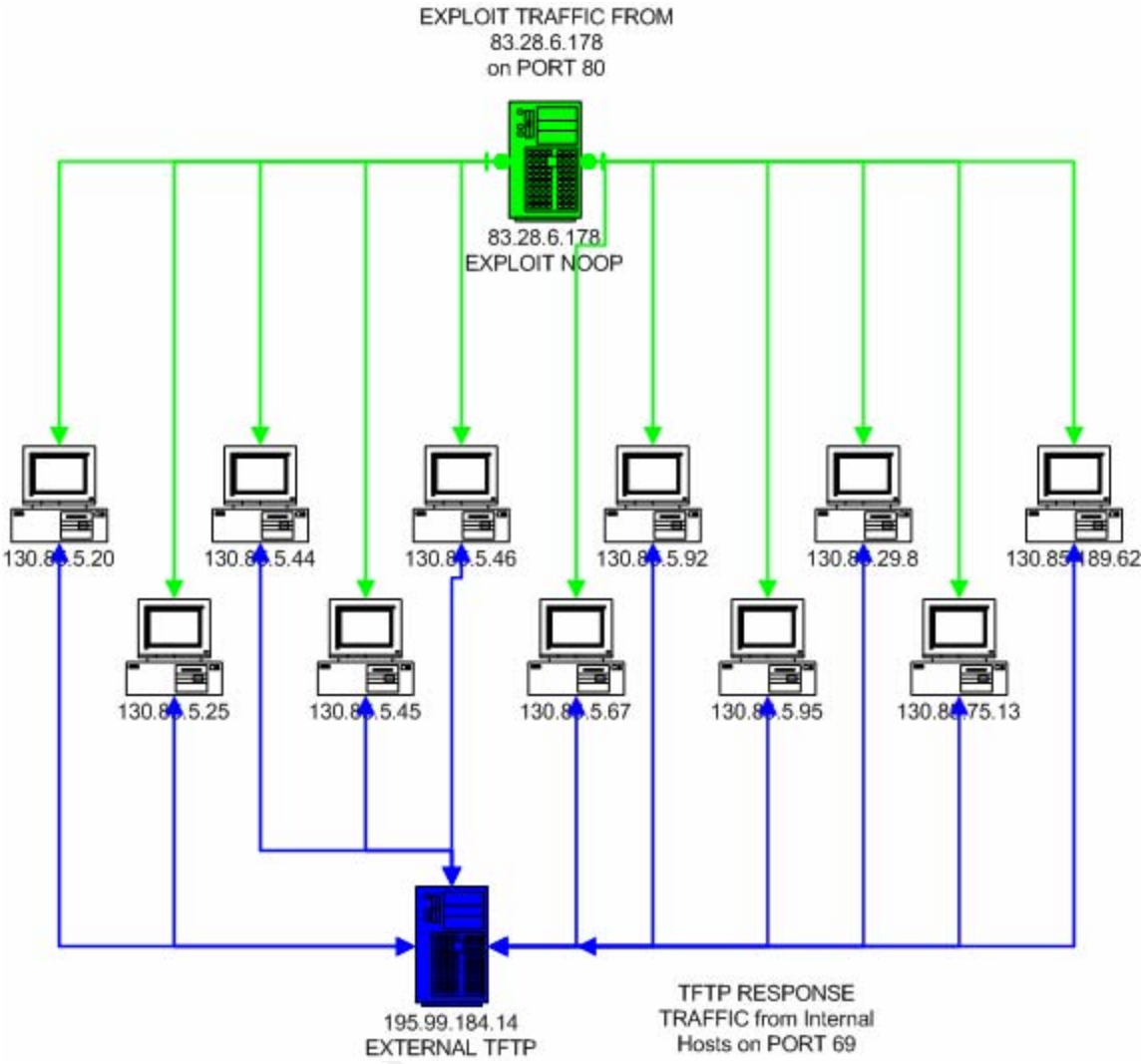
In addition to Offender #3, Offenders #4 is also worthy of further inspection. The 83.28.6.178 address targeted a wide range of machines at port 80, all of whom seemed not to respond in any way to their query. However, 11 of the target machines, after receiving the EXPLOIT x86 NOOP traffic each sent several packets to the 195.99.184.14 host that triggered the TFTP—External TCP connection to Internal TFTP Server rule. This rule is triggered whenever an external or internal

[50] Whitehat Network Security Resource, IDS362
[51] SnortSnarf
[52] UMBC Office of Information Technology

host attempts to make a connection on port 69. Strangely, all of the hosts in question attempted a TFTP connection to the same target address and more importantly, responded to the query on port 69. The following is a sample alert capture to illustrate this communication:

```
02/22-20:20:15.194852 [**] EXPLOIT x86 NOOP [**] 83.28.6.178:55040 -> 130.85.5.45:80
02/22-20:20:15.459682 [**] EXPLOIT x86 NOOP [**] 83.28.6.178:55040 -> 130.85.5.45:80
02/22-20:20:15.804351 [**] EXPLOIT x86 NOOP [**] 83.28.6.178:55040 -> 130.85.5.45:80
02/22-21:39:32.452337 [**] TFTP - External TCP connection to internal tftp server [**] 195.99.184.14:1794 -> 130.85.5.45:69
02/22-21:39:32.452432 [**] TFTP - External TCP connection to internal tftp server [**] 130.85.5.45:69 -> 195.99.184.14:1794
02/22-21:39:32.973529 [**] TFTP - External TCP connection to internal tftp server [**] 195.99.184.14:1794 -> 130.85.5.45:69
02/22-21:39:32.973637 [**] TFTP - External TCP connection to internal tftp server [**] 130.85.5.45:69 -> 195.99.184.14:1794
```

As is evidenced by the traffic above, it would appear that the EXPLOIT attack was successful and in some way caused the targeted host to initiate a TFTP connection to the 195.99.184.14 host when solicited on port 69. As mentioned above, there were 11 machines that followed this pattern, all of which are illustrated in the link diagram below:



It is suspected that this traffic is indicative of some type of worm or other virus, though the exact compromise is uncertain without further analysis and full packet captures. It could be a variant of the Blaster worm, which exploits an RPC vulnerability then listens on port 69 (Blaster can also tunnel the RPC vulnerability over HTTP channels, which may explain the port 80 traffic). Immediate further investigation and (if necessary) sanitation of these machines is recommended. As well, all internal hosts that established a TFTP connection to an external address should be investigated.

Top 10 Talkers:

The following are lists of the Top Ten talkers in terms of Alerts, Scans, and OOS files. The top talkers are defined by those source and destination addresses which generated the most log traffic on the network in their respective areas during this five day period. The top ten talkers lists were generated using SnortSnarf and SawMill. [53]

[53] Sawmill

ALERTS: Top 10 Source Hosts

Rank	Source IP	Total # Alerts	# Signatures triggered	Destinations involved
1	210.98.224.82	34662 alerts	7 signatures	(14311 destination IPs)
2	130.85.29.3	3658 alerts	3 signatures	(4 destination IPs)
3	169.200.215.36	3486 alerts	1 signatures	130.85.29.3
4	68.50.102.64	3378 alerts	1 signatures	130.85.30.4
5	130.85.11.7	3140 alerts	2 signatures	169.254.0.0
6	130.85.21.67	1782 alerts	1 signatures	(8 destination IPs)
7	12.21.173.176	1712 alerts	2 signatures	130.85.30.3, 130.85.30.4
8	130.85.21.69	1568 alerts	3 signatures	(8 destination IPs)
9	130.85.21.68	1482 alerts	1 signatures	(8 destination IPs)
10	68.55.178.168	1468 alerts	2 signatures	130.85.30.3, 130.85.30.4

ALERTS: Top 10 Destination Hosts

Rank	Source IP	Total # Alerts	# Signatures triggered	Destinations involved
1	64.136.21.233	19232 alerts	13 signatures	(17036 source IPs)
2	130.85.30.4	11305 alerts	1 signatures	(299 source IPs)
3	130.85.30.3	4587 alerts	2 signatures	(173 source IPs)
4	169.200.215.36	3655 alerts	1 signatures	130.85.29.3
5	130.85.29.3	3490 alerts	3 signatures	(3 source IPs)
6	169.254.0.0	3348 alerts	2 signatures	(3 source IPs)
7	68.6.96.171	1317 alerts	1 signatures	130.85.153.37
8	69.14.170.227	1138 alerts	1 signatures	(3 source IPs)
9	211.27.66.7	864 alerts	2 signatures	(3 source IPs)
10	24.116.186.172	659 alerts	2 signatures	(3 source IPs)

SCANS: Top 10 Source Hosts

	Source host	Hits	Hits bar
1	130.85.1.3	2,231,711	
2	130.85.1.4	522,253	
3	130.85.81.39	426,934	
4	130.85.111.197	402,797	
5	130.85.84.164	301,114	
6	130.85.70.164	239,325	
7	130.85.34.14	146,943	
8	130.85.80.224	118,958	
9	210.98.224.82	81,232	
10	130.85.82.15	73,771	
	1792 other items	1,435,928	
	Total	**5,980,966**	

SCANS: Top 10 Destination Hosts

	Destination host	Hits	Hits bar
1	192.26.92.30	66,692	
2	69.6.68.10	44,380	
3	192.5.6.30	42,449	
4	192.48.79.30	40,908	
5	69.6.68.11	35,110	
6	203.20.52.5	34,898	
7	130.85.97.43	28,424	
8	216.109.116.17	23,998	
9	130.85.25.73	23,715	
10	131.118.254.33	22,352	
	992085 other items	5,618,040	
	Total	**5,980,966**	

OOS: Top 10 Source Hosts

Host	#Alerts (Tot.)
61.109.209.144	10927
68.54.84.49	1143
81.9.192.27	291
147.27.1.193	186
82.161.49.116	137
66.225.198.20	125
67.114.19.186	89
217.122.200.151	86
205.252.97.14	77
212.202.77.157	69

OOS: Top 10 Destination Hosts

Host	#Alerts (Tot.)
130.85.6.7	1206
130.85.12.6	614
130.85.24.44	340
130.85.84.235	227
130.85.42.10	187
130.85.24.34	168
130.85.42.13	131
130.85.42.11	90
130.85.70.164	74
130.85.42.6	69

Detailed Analysis--Five Top Suspects:

The following five hosts were selected for further analysis based on the number of alerts they generated, as well as the severity.

HOST: 130.85.1.3
DNS Name: umbc3.umbc.edu
Owner: UMBC
Type: Internal
Correlation: Number 1 Scanner in Scans files
Description:

This host is an internal DNS server that generated unusual amounts of scan data, due to the fact that it handles so many requests. This was suspicious simply for the sheer amount of alerts generated, though the alerts are false positives and are the result of a poorly tuned portscan preprocessor in the NIDS.

HOST: 130.85.30.4
DNS Name: lan2.umbc.edu
Owner: UMBC
Type: Internal
Correlation: Target in Top 10 list of alerts generated.
Description: Generated large amounts of alert data, due to a Snort rule that is too broad and captures all traffic directed to this IP.

HOST: 210.98.224.82
DNS Name: Unknown
Owner: Booil Mobile Telecom Corporation
Type: External
Correlation: Number 1 SUNRPC scanner source address.
Description: This host scanned over 14,000 hosts on the internal network, presumably looking for an RPC vulnerability. Not only does this make it suspicious enough to investigate further, but the abuse representative for this domain should also be notified.

HOST: 61.109.209.144
DNS Name: Unknown
Owner: Korea Network Information Center
Type: External
Correlation: Number 1 Out of Spec packet generator
Description: Generated the largest number of OOS packets in the OOS logs. This host scanned over 10,000 hosts on the internal network for port 21 (FTP) traffic. As in the case of the SUNRPC scanner above, the abuse representative for the domain should be contacted and informed of this activity. If policy permits, this source address should be blocked at the perimeter.

HOST: 83.28.6.178
DNS Name: jg178.neoplus.adsl.tpnet.pl
Owner: Neostrada-ADSL
Type: External
Correlation: One of the Top 5 EXPLOIT NOOP alert generators.
Description: Generated suspicious EXPLOIT x86 NOOP traffic, which seemed to compromise 11 hosts on the network (see Alert #9 Analysis above for more information).

Correlations:

The following GIAC submissions were used for correlation during the research and writing of this paper:

Ian Martin: http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf

Pete Storm: http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf

Ricky Smith: http://www.giac.org/practical/GCIA/Ricky_Smith_GCIA.pdf

Marshal Heilm: http://www.giac.org/practical/GCIA/Marshall_Heilman_GCIA.pdf

As well, the ARIN, RIPE, APNIC, DShield, and Port Database features of SnortSnarf were heavily used for correlation and host identification. See References for complete source information.

Defensive Recommendations:

Though this network has been well secured by the University Administration team, there are a few points of improvement that can still be made. They are as follows:

- Many of the rules mentioned in the Alert Detection Analysis section need to be tuned in order to reduce the amount of unnecessary log data generated. Regular audits of the NIDS ruleset should be performed in order to ensure that the NIDS is keeping up with an ever-changing network environment. If more detailed Intrusion Detection is required on certain hosts, a Host-based IDS installation should be implemented on an individual basis.

- Any servers and/or services that should not be accessed from an external source (such as lan1 and lan2) should have access restricted at the firewall or border device.

- In addition to these general security recommendations, there are several hosts on the network that appear to have been compromised—these hosts (listed above in Alert #9) should be immediately examined and sanitized.

- Though not mentioned above, the novell.umbc.edu web server should have its Apache banner modified and/or sanitized (if it hasn't been already). When a banner grab is performed, the exact version of Apache is given (Apache / 1.3.27 [Netware] mod_jk/1.2.2_dev).

- Since the University appears to be using an older version of Snort, it is recommended that the NIDS be upgraded to the latest version (Snort 2.1.1). In addition, the preprocessors included with Snort should be tuned for the University environment (for example, to accommodate the large amounts of DNS data that are present on the network) in order to reduce the number of false positives generated and provide more useable log data.

Description of Analysis Process:

The tools used in the analysis of this data included SnortSnarf and various Unix utilities (ported for Windows) such as grep and sed. All of the respective log files were merged into three files for correlation (alerts.ids, scans.ids, and oos.ids). This made analysis especially difficult due to the size and quantity of log information, and "as is" the data was too large to be processed in a reasonable amount of time by SnortSnarf. In order to break up the data into more manageable sections, the portscan data was removed from the alert log files before being input into SnortSnarf

(portscan data is also included in the scans log files, which were examined using grep and Sawmill, a web-based database reporting program). In addition, the "MY.NET" entries in the alerts and OOS files were changed to 130.85, using the IP scheme found in the scans files. This allowed for quicker data analysis, since SnortSnarf was then able to process the data, as well as perform Whois and DShield lookups on the IP addresses.

In order to parse the OOS files for use with SnortSnarf, Ricky Smith's "parse-oos.pl" script was used (see Correlations). Microsoft Excel and its sorting feature was also used in order to find patterns and groups within source and destination IPs (though it is only useful with smaller amounts of data).

Some samples of the commands used for processing data:

To replace "MY.NET" with "130.85" in the OOS file:
```
sed "s/MY.NET/130.85/g" c:\sans\analyze\oos\oos.ids >> oos1.ids
```

SnortSnarf command used to process alert file in Windows 2000 (all on one line):
```
snortsnarf.pl -d c:\Inetpub\wwwroot\log -db c:\snortsnarf\ann-
dir\annotation-base.xml -cgidir c:\inetpub\wwwroot\cgi
c:\downloads\alert.ids
```

Output all lines except for those which contain this IP address and output to file:
```
grep -v "210.98.221.82" sunrpc3.ids >> sunrpc4.csv
```

Replace "->" with "," using sed and output to file (for use in a CSV file or Spreadsheet):
```
sed "s/\->/,/g" alert.ids >> alert2.ids
```

References: Analyze This

F-Secure. "FSecure Virus Descriptions: Adore." April 2001. URL: http://www.f-secure.com/v-descs/adore.shtml

LURHQ Threat Intelligence Group. LURHQ Homepage. URL: http://www.lurhq.com/idsindepth.html.

Martin, Daniel. "Re: Spoofed SMB name wildcard probes." Online posting. Incidents.org. 4 May 2001. URL: http://cert.uni-stuttgart.de/archive/incidents/2001/05/msg00041.html

Microsoft Corporation. "Windows XP Home Edition Product Documentation." URL: http://www.microsoft.com/windowsxp/home/using/productdoc/en/default.asp?url=/windowsxp/home/using/productdoc/en/sag_tcpip_pro_autoconfig.asp

Roesch, Martin. "Re: [Snort-users] Incomplete Packet Fragments Discarded." Online posting. Snort Users List. 26 November 2001. URL: http://www.mcabee.org/lists/snort-users/Nov-01/msg00820.html

SawMill. Sawmill Homepage. URL: www.sawmill.net

UMBC Office of Information Technology. "UMBC Security Updates: Internet Worm Phatbot and the Witty Worm." URL: http://www.umbc.edu/oit/article.html?news_id=136

Whitehats Network Security Resource. "IDS177—'NETBIOS-NAME-QUERY." arachNIDS Intrusion Detection Database.
URL: http://www.whitehats.com/info/IDS177

Whitehats Network Security Resource. "IDS362—'SHELLCODE-X86-NOPS-UDP." arachNIDS Intrusion Detection Database.
URL: http://www.whitehats.com/info/IDS362

Whitehats Network Security Resource. "Search for: rpc 32771." arachNIDS Intrusion Detection Database.
http://www.whitehats.com/cgi/arachNIDS/Search?search=rpc+32771

Last Updated: May 25th, 2017

Upcoming Training

Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC503: Intrusion Detection In-Depth	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS vLive - SEC503: Intrusion Detection In-Depth	SEC503 - 201709,	Sep 11, 2017 - Oct 18, 2017	vLive
Baltimore September 2017 - SEC503: Intrusion Detection In-Depth	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Scottsdale SEC503	Scottsdale, AZ	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Boston SEC503	Boston, MA	Oct 09, 2017 - Oct 14, 2017	Community SANS
Community SANS Ottawa SEC503	Ottawa, ON	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced