Crist Clark - GCIA Practical Assignment

GIAC Intrusion Detection Curriculum
Network Security 2000
Monterey, CA
October 15-20, 2000

Contents

Network Detect #1: Napster Strikes Back

```
30Oct2000  8:15:24 accept >qfe3  tcp XXX.XXX.153.34:2263 -> 128.100.71.72:6699
30Oct2000  8:15:29   drop >hme0  tcp 128.100.71.72:255 -> XXX.XXX.248.142:6699
30Oct2000  8:17:06   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.153.34:2263
30Oct2000  8:19:50   drop >hme0  tcp 128.100.71.72:http -> XXX.XXX.248.142:669
30Oct2000  8:20:20   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.153.34:2263
30Oct2000  8:24:09   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.153.34:2263
30Oct2000  8:29:17 accept >qfe3  tcp XXX.XXX.153.34:2267 -> 128.100.71.72:6699
30Oct2000  8:29:25 accept >qfe3  tcp XXX.XXX.153.34:2268 -> 128.100.71.72:6699
30Oct2000  8:31:57   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.153.34:2263
30Oct2000  8:32:10   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.248.142:234
30Oct2000  8:32:40   drop >hme0  tcp 128.100.71.72: -> XXX.XXX.248.142:6699 40
30Oct2000  8:34:36   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.153.34:2263
30Oct2000  8:34:53   drop >hme0  tcp 128.100.71.72:daytime -> XXX.XXX.248.142:
30Oct2000  8:38:05 accept >qfe3  tcp XXX.XXX.153.34:2270 -> 128.100.71.72:6699
30Oct2000  8:44:05   drop >hme0  tcp 128.100.71.72:208 -> XXX.XXX.248.142:6699
30Oct2000  8:46:42   drop >hme0  tcp 128.100.71.72:6 -> XXX.XXX.248.142:6699 4
30Oct2000  8:49:16   drop >hme0  tcp 128.100.71.72:42 -> XXX.XXX.248.142:6699
30Oct2000  8:50:42   drop >hme0  tcp 128.100.71.72:6699 -> XXX.XXX.248.142:295
30Oct2000  8:52:31 accept >qfe3  tcp XXX.XXX.153.34:2273 -> 128.100.71.72:6699
```

Source of Trace

Our company's Internet firewall which protects several class C networks.

Detect Generated by

Custom AWK scripts applied to "exported" Checkpoint Firewall-1 logs. The scripts and commands used to generate the logs from FW-1 appear in Appendix A. The format of the logs is fairly simple: date, time, firewall action, in (>) or out (<) interface, IP protocol, source address and port, destination source and port, and packet size. The second pair of source and destination addresses in parenthenses are translated NAT values, that is, the numbers seen by the outside world. qfe3 is the interior interface and hme0 is external.

Like many firewalls, the log information is very bare bones. We do not have details about TCP flags, TCP options, IP TTL, IP ID, etc. But we have to make a go with what is available.

Probability Source Address Spoofed

Very unlikely. Unless a man-in-the-middle or eavesdropper decided to spoof when he saw the connection attempt, the chance that someone coincidentally spoofed that address at the same time our user tried to connect is minute.

Description of Attack

After our user initially connects to the external server, 128.100.71.72, we start to see some very interesting packets coming back. The port 6699 is typically associated with Napster and other file sharing protocols. Seeing some connections trying to come back in once a Napster session has started is not too unusual, but there are some interesting features that are further discussed in the following sections.

Attack Mechanism

After the initial connection attempt by our user (we have no way to know if it is successful or not), we begin to get a slow scan coming back. Notice that NAT has caused the remote attacker to probe our firewall rather than the true source. What makes this particular collect interesting are the unusual source ports on the returning connections and inferences we can make about the packets due to their size.

Since we lack TCP flag information, it is hard to say why the packets that look like they be part of the connection established by our internal user are being dropped. But note that the returning packets are 40 bytes. It is not unusual to see lost, out of sequence, or otherwise plain broken packets bouncing off of the firewall from Napster traffic due to the unreliable nature of the net. However, such packets returning from a Napster server are typically bloated with data, often full 1500 byte packets (maximum size we can push through the Ethernet outside the firewall). Here packets are not carrying data, but only carrying information for the TCP connection. They may be ACK packets, but that would imply our client is pushing data which is either unlikely or very bad. We do not want people sending data out of our network.

What makes this particular collect look less like a broken TCP packets from an every-day

Napster connection and more like an attack are the source ports of the incoming connections. In particular, the source port of the second connection attempt is HTTP (port 80). This wreaks of an attempt to circumvent a stateless packet filter. Look at the other source ports, 255, daytime (13), 208, 6, and 42. And there is the puzzling blank port. I am not sure what the firewall is trying to tell me with that (perhaps port 0?), but it is really what the log has and not a problem with the scripts. We would expect "ephemeral" ports to be used for incoming connections and analysis of other Napster sessions would seem to confirm this. This is what makes me consider this trace a possible attack.

Correlations

The source address has a reverse lookup of,

```
;; ANSWER SECTION:
72.71.100.128.in-addr.arpa.  23h41m56s IN PTR  helicotrema.ibme.utoronto.ca.
```

And a quick check with ARIN (whois.arin.net) verifies that it is indeed within the University of Toronto block. From the name, helicotrema (part of the inner ear), it definitely belongs to the Institute of Biomedical Engineering (IBME). However, university computers are notorious for being abused by students who may or may not have legitimate access as well as compromise by script kiddies. Little can be determined from the identity of the source. No other traffic from our network was observed to this address. There is no listener on port 6699 of the attacking host at the time of this writing.

I mentioned several times what "typical" Napster traffic looks like. Below is an example,

```
2Nov2000  9:31:35 accept >qfe3  tcp 192.168.XXX.186:1412 -> 172.144.30.193:66
2Nov2000  9:31:44 accept >qfe3  tcp XXX.XXX.152.239:1812 -> 203.96.106.137:66
2Nov2000  9:35:44   drop >hme0  tcp 200.28.48.106:4020 -> XXX.XXX.248.142:669
2Nov2000  9:41:25   drop >hme0  tcp 62.36.149.102:1104 -> XXX.XXX.248.142:669
2Nov2000  9:44:12   drop >hme0  tcp 198.213.203.57:1133 -> XXX.XXX.248.142:66
2Nov2000  9:44:27   drop >hme0  tcp 212.120.103.108:1273 -> XXX.XXX.248.142:6
2Nov2000  9:47:51   drop >hme0  tcp 195.223.93.163:1185 -> XXX.XXX.248.142:66
2Nov2000  9:52:21 accept >qfe3  tcp XXX.XXX.153.168:1292 -> 62.227.192.50:669
2Nov2000  9:56:31 accept >qfe3  tcp XXX.XXX.153.196:1148 -> 4.4.58.52:6699 44
2Nov2000 10:02:32   drop >hme0  tcp 63.16.57.29:1180 -> XXX.XXX.248.142:6699
2Nov2000 10:10:08   drop >hme0  tcp 212.14.119.159:1458 -> XXX.XXX.248.142:66
```

Internal users make connections to external machines and suddenly we get pounded by multiple sites. But notice that the connections are not as persistent as above nor do they have the suspicious source ports. A more thorough analysis for all Napster traffic during the week shows similar patters as this example, although this particular snapshot is heavy on incoming attempts relative to outgoing. In actuality, there were more than nine outgoing connections for every attempt to come in (911 to 98) for the week of data checked. In addition, no other external hosts made multiple connection attempts from more than one source port.

When initially doing this analysis, I did not have any other examples of this type of scan, and as I previously mentioned, my data is quite limited with regard to things like TCP flags or IP parameters like TTL and ID. However, while working on the Analyze This portion of the

assignment, I found *lots* of really strange stuff coming in on port 6699 in the data sets. A few examples,

```
Aug 17 08:43:01 130.239.11.230:0 -> MY.NET.181.173:6699 NOACK 2*SFR**U RE
Aug 17 08:43:32 130.239.11.230:6699 -> MY.NET.181.173:4554 NULL ********
Aug 17 08:43:38 130.239.11.230:0 -> MY.NET.181.173:6699 NOACK ****R**U
Aug 17 08:43:58 130.239.11.230:6699 -> MY.NET.181.173:4554 INVALIDACK **S
Aug 17 08:45:32 130.239.11.230:6699 -> MY.NET.181.173:4555 UNKNOWN 21**R*

Sep  4 12:56:39 161.184.104.111:6699 -> MY.NET.222.154:1883 UNKNOWN *1***
Sep  4 12:57:23 161.184.104.111:6699 -> MY.NET.222.154:1883 INVALIDACK **
Sep  4 12:59:14 161.184.104.111:1 -> MY.NET.222.154:6699 INVALIDACK ***FR

Sep  4 16:27:38 130.234.185.71:6699 -> MY.NET.221.170:1243 NULL ********
Sep  4 16:27:45 130.234.185.71:6699 -> MY.NET.221.170:1243 NULL ********
Sep  4 16:29:26 130.234.185.71:0 -> MY.NET.221.170:6699 NOACK *1**RP*U RE
Sep  4 16:30:29 130.234.185.71:86 -> MY.NET.221.170:6699 NOACK *1**RP*U R
```

That pattern looks an awful lot like what I saw. In these examples, we see invalid packets coming back on port 6699, null packets or other invalid flag combinations. Perhaps that is why my firewall was dropping incoming packets. The strange blank source port in my data probably corresponds to the 0 source port in these captures, a bug in the FW-1 logging code.

These are starting to look very much like fingerprinting and attempts to circumvent dumb packet filters. The remote machine is sending invalid packets to see if they make it to the remote machine (pass any firewalls) and then see how the machine responds (for the fingerprint).

Evidence of Active Targeting

From the deductions of the previous sections, it appears we were actively targeted. The trigger was the Napster connection from one of our hosts to the attacker. The multiple connections attempts from multiple and suspicious source ports coupled with the lack of this behavior from other Napster servers leads to the conclusions that this was likely a reconnaissance scan of some type. From the correlations with the more detailed Snort data from the Analyze This data we were provided, this looks very much like a fingerprinting attempt.

Severity

Target Criticality:	2	The target was a Windows NT desktop system. The systems typically contain little data (network drives store important things) and can easily be replaced or repaired.
Lethality:	2	The lethality is harder to judge since we do not know what the attacker was trying to do. I have concluded that it was most probably a fingerprinting and firewall evading attempt. But what would the attacker do next if he successfully reached and IDed the machine? Does he have a platform specific exploit for Napster? That would explain the interest in connecting back on 6699. Maybe he just really wants to connect to our Napster and check out our music. Or maybe he has a completely different exploit in mind if he gets through? Is he just IDing remote hosts with no

		malicious intent? Who knows, maybe he's collecting marketing data. All of that considered, let us just consider the scan itself. It was not particularly dangerous.
System Countermeasures:	- 1	Our internal user is running Napster software. It has had security problems in the past (CAN-2000-0281, CAN-2000-0412). However, even if the software has vulnerabilities, it should be running as a non-privileged user.
Network Countermeasures:	- 3	Incoming attempts were blocked at the firewall and our use of NAT protected the true address of our internal host. However, Napster is a two way sharing protocol and we have reason to suspect there may have been a flow of data out. The difficultly of determining the intent of the attacker lowers our network protection. Assuming FW-1 locked tight is close to a 5, knock off a point for letting people Napster out and another point for the weak logging at my disposal.
Severity:	0	The total threat from the attack is low.

Defensive Recommendation

The firewall did its job, even if the logging leaves something to be desired. Blocking Napster going out is a possibility being considered. Presently, it is tolerated so long as network performance is not hurt. Total bandwidth usage leaving our network is monitored. This can prevent large transfers from leaving the network unnoticed. If they are observed, manual action can be taken to determine the source.

Multiple Choice Question

In the first detect above, even though we lack TCP flag information, which one of the following is the LEAST likely possibility for the recurring, dropped TCP traffic coming from 128.100.71.72:6699 to XXX.XXX.153.34:2263,

a. SYN connection attempts
b. ACK-PSH retries
c. SYN-ACK retries
d. SYN-FIN scans

ANSWER: b. ACK-PSH packets should contain data. The returning packets are small; the same size as packets coming in to establish a connection. They do not carry data. All of the other choices are more likely possibilities.

Return to top.

Network Detect #2: SOCKS, Telnet, and a little IRC

```
 9Oct2000  9:14:05   drop >hme0  tcp 203.101.17.225:41095 -> XXX.XXX.248.142:S
 9Oct2000  9:14:05   drop >hme0  tcp 203.101.17.225:41096 -> XXX.XXX.248.142:t
12Oct2000  8:15:11   drop >hme0  tcp 203.101.17.225:45176 -> XXX.XXX.248.142:S
```

```
12Oct2000  8:15:11   drop >hme0  tcp 203.101.17.225:45177 -> XXX.XXX.248.142:t
17Oct2000 15:04:48   drop >hme0  tcp 203.101.17.225:34127 -> XXX.XXX.248.142:S
17Oct2000 15:04:48   drop >hme0  tcp 203.101.17.225:34128 -> XXX.XXX.248.142:t
18Oct2000  8:44:53   drop >hme0  tcp 203.101.17.225:55267 -> XXX.XXX.248.142:S
18Oct2000  8:44:53   drop >hme0  tcp 203.101.17.225:55268 -> XXX.XXX.248.142:t
20Oct2000 10:11:09   drop >hme0  tcp 203.101.17.225:56599 -> XXX.XXX.248.142:S
20Oct2000 10:11:09   drop >hme0  tcp 203.101.17.225:56600 -> XXX.XXX.248.142:t
20Oct2000 13:32:29   drop >hme0  tcp 203.101.17.225:47415 -> XXX.XXX.248.142:S
20Oct2000 13:32:29   drop >hme0  tcp 203.101.17.225:47416 -> XXX.XXX.248.142:t
30Oct2000 10:59:05   drop >hme0  tcp 203.101.17.225:41623 -> XXX.XXX.248.142:S
30Oct2000 10:59:05   drop >hme0  tcp 203.101.17.225:41624 -> XXX.XXX.248.142:t
30Oct2000 13:47:19   drop >hme0  tcp 203.101.17.225:50625 -> XXX.XXX.248.142:S
30Oct2000 13:47:19   drop >hme0  tcp 203.101.17.225:50626 -> XXX.XXX.248.142:t
31Oct2000 10:24:52   drop >hme0  tcp 203.101.17.225:57006 -> XXX.XXX.248.142:S
31Oct2000 10:24:52   drop >hme0  tcp 203.101.17.225:57007 -> XXX.XXX.248.142:t
31Oct2000 14:42:00   drop >hme0  tcp 203.101.17.225:45119 -> XXX.XXX.248.142:S
31Oct2000 14:42:00   drop >hme0  tcp 203.101.17.225:45120 -> XXX.XXX.248.142:t
31Oct2000 14:46:06   drop >hme0  tcp 203.101.17.225:45371 -> XXX.XXX.248.142:S
31Oct2000 14:46:06   drop >hme0  tcp 203.101.17.225:45372 -> XXX.XXX.248.142:t
 1Nov2000  9:12:45   drop >hme0  tcp 203.101.17.225:48972 -> XXX.XXX.248.142:S
 1Nov2000  9:12:45   drop >hme0  tcp 203.101.17.225:48973 -> XXX.XXX.248.142:t
 2Nov2000 13:10:32   drop >hme0  tcp 203.101.17.225:34516 -> XXX.XXX.248.142:S
 2Nov2000 13:10:32   drop >hme0  tcp 203.101.17.225:34517 -> XXX.XXX.248.142:t
 3Nov2000 10:03:42   drop >hme0  tcp 203.101.17.225:39692 -> XXX.XXX.248.142:S
 3Nov2000 10:03:42   drop >hme0  tcp 203.101.17.225:39693 -> XXX.XXX.248.142:t
 3Nov2000 13:49:32   drop >hme0  tcp 203.101.17.225:56618 -> XXX.XXX.248.142:S
 3Nov2000 13:49:32   drop >hme0  tcp 203.101.17.225:56619 -> XXX.XXX.248.142:t
```

Source of Trace

Our company's Internet firewall which protects several class C networks.

Detect Generated by

Custom AWK scripts applied to "exported" Checkpoint Firewall-1 logs. See the description in Detect #1 for details.

Probability Source Address Spoofed

Unlikely. These look like real TCP connection attempts.

Description of Attack

The attacker is repeatedly probing us for open telnet (23) and SOCKS (1080) ports.

Attack Mechanism

The attacker tries a SOCKS connection and then a telnet in rapid succession.

Correlations

The source, 203.101.17.225, has a reverse lookup of,

```
;; ANSWER SECTION:
225.17.101.203.in-addr.arpa.  1H IN PTR  irc.one.net.au.

;; AUTHORITY SECTION:
17.101.203.IN-ADDR.ARPA.  1H IN NS  red.one.net.au.
17.101.203.IN-ADDR.ARPA.  1H IN NS  orange.one.net.au.

;; ADDITIONAL SECTION:
red.one.net.au.         15M IN A        203.17.224.11
orange.one.net.au.      15M IN A        203.17.224.12
```

The APNIC database (whois.apnic.net) verifies that the source address falls within the block of One Net which appears to be an Australian ISP. The source address, their IRC server, seems somewhat ominous. It may be a rooted box. Then again, I have heard of IRC servers behaving like this as an attempt to check for people anonymizing themselves with open SOCKS proxies.

What was particularly worrisome in this case was that I was at first unable to correlate the incoming attempts with any outgoing activity. In Detect #1, it seemed clear that the incoming attempts were prompted by a particular internal machine making an outgoing connection. In this case, no internal machines have tried to access this external IRC server during the time frame of the incoming connection attempts.

This made me wonder if I had a trojan or some other malware on our internal net which was "phoning home" on IRC. Perhaps a listening agent is connected to an IRC channel somewhere. When an infected box connects to the channel, the agent tries to SOCKS and telnet to the infected machine. This agent could easily be listening on an different IRC server from the one the infected machine calls up. It is a scary thought and one not easy to confirm or deny. I went through the logs dumping the connections immediately before the scans, but was not able to find a source that correlated with the attacks.

Since IRC seems to be involved by the name of the scanning host, I decided to look at outgoing IRC attempts. A quick examination showed that there was one "promising candidate,"

```
 9Oct2000  9:12:49 accept >qfe3  tcp 172.XX.XXX.20:4752 -> 216.152.64.155:6667
 9Oct2000  9:15:33 accept >qfe3  tcp 172.XX.XXX.20:4754 -> 206.101.197.250:666
12Oct2000  8:10:06 accept >qfe3  tcp 172.XX.XXX.20:1174 -> 206.101.197.250:666
12Oct2000  8:13:55 accept >qfe3  tcp 172.XX.XXX.20:1176 -> 216.152.64.155:6667
17Oct2000 13:55:18 accept >qfe3  tcp 172.XX.XXX.20:1943 -> 206.101.197.250:666
17Oct2000 15:03:32 accept >qfe3  tcp 172.XX.XXX.20:1945 -> 216.152.64.155:6667
18Oct2000  8:41:00 accept >qfe3  tcp 172.XX.XXX.20:1574 -> 206.101.197.250:666
18Oct2000  8:43:38 accept >qfe3  tcp 172.XX.XXX.20:1576 -> 216.152.64.155:6667
20Oct2000 10:09:49 accept >qfe3  tcp 172.XX.XXX.20:2633 -> 216.152.64.155:6667
20Oct2000 13:31:13 accept >qfe3  tcp 172.XX.XXX.20:2764 -> 216.152.64.155:6667
30Oct2000 10:57:49 accept >qfe3  tcp 172.XX.XXX.20:1050 -> 216.152.64.155:6667
30Oct2000 11:02:24 accept >qfe3  tcp 172.XX.XXX.20:1051 -> 216.152.64.155:6667
30Oct2000 11:03:17 accept >qfe3  tcp 172.XX.XXX.20:1052 -> 216.152.64.155:6667
30Oct2000 13:20:36 accept >qfe3  tcp 172.XX.XXX.20:1072 -> 206.101.197.250:666
30Oct2000 13:31:37 accept >qfe3  tcp 172.XX.XXX.20:1074 -> 206.101.197.250:666
30Oct2000 13:46:03 accept >qfe3  tcp 172.XX.XXX.20:1077 -> 216.152.64.155:6667
31Oct2000 10:23:36 accept >qfe3  tcp 172.XX.XXX.20:2940 -> 216.152.64.155:6667
```

```
31Oct2000 14:38:43 accept >qfe3  tcp 172.XX.XXX.20:1083 -> 216.152.64.155:6667
31Oct2000 14:40:44 accept >qfe3  tcp 172.XX.XXX.20:1084 -> 216.152.64.155:6667
31Oct2000 14:44:40 accept >qfe3  tcp 172.XX.XXX.20:1085 -> 216.152.64.155:6667
 1Nov2000  9:11:30 accept >qfe3  tcp 172.XX.XXX.20:1101 -> 216.152.64.155:6667
 2Nov2000 13:05:03 accept >qfe3  tcp 172.XX.XXX.20:1251 -> 216.152.64.155:6667
 2Nov2000 13:06:38 accept >qfe3  tcp 172.XX.XXX.20:1252 -> 216.152.64.155:6667
 2Nov2000 13:07:29 accept >qfe3  tcp 172.XX.XXX.20:1253 -> 216.152.64.155:6667
 2Nov2000 13:08:59 accept >qfe3  tcp 172.XX.XXX.20:1255 -> 198.165.100.11:6667
 3Nov2000 10:02:11 accept >qfe3  tcp 172.XX.XXX.20:4922 -> 198.165.100.11:6667
 3Nov2000 13:48:01 accept >qfe3  tcp 172.XX.XXX.20:4994 -> 198.165.100.11:6667
```

Just a little to close to be coincidental, no? So let me do a quick NMap scan of the source from inside my network,

```
$ nmap 172.XX.XXX.20

Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on  (172.XX.XXX.20):
(The 1520 ports scanned but not shown below are in state: closed)
Port       State       Service
139/tcp    open        netbios-ssn
12345/tcp  open        NetBus
31337/tcp  open        Elite

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

Eiiieeeee! Not to be too dramatic, but I immediately hunted down the machine. It turns out that the user was running some software on his PC called LockDown 2000(tm). It is a firewall/honeypot package. From what I could deduce from the documentation, it should not go as far as chat on IRC to entice script kiddies to bang on it (although I did not see mention of the fact that it phoned home on HTTP periodically but did see it in the logs), but my NMap scan came up positive since it listens on common trojan ports for scan attempts.

I had to wait until the next day, but I got hold of the user of the machine in question. He reported that he uses IRC. So after the scare above, it looks like all we have some kind of 'bot listening on IRC that scans users' source IP addresses. It is most likely a defensive measure by the IRC servers.

Evidence of Active Targeting

We were definitely being targeted. The pattern repeated regularly over a long period to a single IP address; the source address that is on outgoing NATed traffic. It was almost certainly someone backtracing the source IP of some outgoing traffic.

Severity

The target machine, the notebook PC, was never listening on telnet or SOCKS ports. The attacks themselves may have not been hostile, but a security check from the server looking out for kiddies using open SOCKS proxies as anonymizers.

Target Criticality: 2 The target was a notebook PC.

Lethality:	1
System Countermeasures:	-3 The system was running security software. However, in quick research, I have found reports that the firewall/honeypot software is not terribly strong. The system is a Win9x machine with minimal services (for Win9x) running and anti-virus software was properly configured.
Network Countermeasures:	-4 NAT misdirected the connect-back attempts and the firewall easily blocked them. The only drawback again is the weak logging.
Severity:	-4 A negative value says there is essentially no risk to the system. However, this number would be different if I had not put the effort into tracking down the attack and checking up that "legit" IRC servers really do scan remote hosts for defensive purposes.

Defensive Recommendation

Our Internet firewall was doing its job and blocking the attempts. I discussed the firewall/honeypot software with the user, discontinued its use, and de-installed it. I have also sent mail to the source of the probes to see if they will verify that the probes are not malicious. In addition, the DNS records of the IRC server that our user was connecting to (remember it is not the same one the probes come from) are suspicious. It very well may be a hijacked box. I contacted the administrators of that system as well.

Multiple Choice Question

IRC has a reputation of being a playground for script kiddies and other unsavory characters. Administrators of "legit" IRC servers have gone so far as to automate scans of incoming hosts' SOCKS and telnet ports in an attempt to:

a. Make sure the client system is not running distributed denial of service attack tools.
b. Check that the connection is not too lossy or lagged.
c. Verify that the user is not using an open proxy or rooted box to hide his true location.
d. Determine if the TCP session is hijacked (Mitnick attack).

ANSWER: c. From responses I received after inquiring about this practice on INCIDENTS@securityfocus.com this is a fairly widespread practice among IRC administrators. The SOCKS check is for open proxies and the telnet for rooted boxes with password-less logins. One person mentioned that the fact the scan actually comes from a different host than the connection went to is a feature to detect software that drops reflected connections.

Return to top.

Network Detect #3: The 10101 Tool

(Note: The first scan shown below was edited edited for length. The "..." represent a continuing scan to the end of the Class C block.)

```
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.0:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.1:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.2:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.4:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.5:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.6:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.7:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.8:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.9:su
16Oct2000  7:14:46   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.152.10:s
...
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.0:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.1:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.2:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.4:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.5:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.6:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.7:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.8:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.9:su
16Oct2000  7:14:47   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.153.10:s
...
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.0:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.1:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.2:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.4:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.5:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.6:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.8:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.9:su
16Oct2000  7:14:48   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.154.10:s
...
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.0:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.1:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.2:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.4:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.5:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.6:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.7:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.8:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.9:su
16Oct2000  7:14:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.155.10:s
...
16Oct2000  7:16:49   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.141:
16Oct2000  7:16:49   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.144:
16Oct2000  7:16:49   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.145:
16Oct2000  7:16:49   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.146:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.148:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.149:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.150:
...
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.188:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.189:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.190:
16Oct2000  7:16:50   drop >hme0  tcp 205.251.216.241:10101 -> XXX.XXX.248.191:

17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.192:sunrp
```

```
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.193:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.194:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.195:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.196:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.197:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.198:sunrp
17Oct2000 14:16:48   drop >hme0  tcp 12.7.184.17:10101 -> YYY.YYY.31.199:sunrp

 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.192:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.193:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.194:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.195:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.196:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.197:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.198:sun
 3Nov2000  5:59:58   drop >hme0  tcp 24.20.135.187:10101 -> YYY.YYY.31.199:sun
```

Source of Trace

Our company's Internet firewall which protects several class C networks.

Detect Generated by

Custom AWK scripts applied to "exported" Checkpoint Firewall-1 logs. See the description in Detect #1 for details.

Probability Source Address Spoofed

Unlikely, but in some cases it could be possible. These look like a SYN-scan (possibly SYN-FIN or other "stealthy" scan, more in the correlations section below) from an special scanning tool. The attacker would need to hear responses to gather information.

Description of Attack

Shown above are three separate scans from three IP addresses on three different days. All are scans for the portmap service, port 111. These are most probably scans from different individuals using the same, specialized scanning tool, but it could possibly be one person or a small group per persons using the same tool from lots of sacrificial, rooted boxes.

Attack Mechanism

The scanner is looking for machines with the portmapper service running. This is a classic. RPC services, for which the portmapper plays an essential role, have had countless exploitable holes on a wide variety of platforms and operating systems (the CVE list is too long to be included here). Since none of our scans returned a favorable response to our kiddies, we did not get to personally see what holes he may have been actually trying to exploit, but others did as we see in the next section.

Note that in all cases the source port, 10101, does not change. This is the obvious signature of a

specialized tool. Normal connection attempts, those going through a normal operating system, will have a changing source port. Typically, we see the source port incrementing +1 each connection.

Correlations

As stated, this is clearly coming from a scanning tool. That we see it used several times over a few days, and that we have not seen this signature previously, is interesting. Before I had a chance to correlate these scans, I had already seen chatter from other security administrators on the INCIDENTS@securityfocus.com mail list about this tool. Let's see what some of the others had found.

The thread started with Abe Getchell at the Kentucky Department of Education who reported in his mail,

```
Subject: Network Scan - sunrpc
Date: Mon Oct 30 2000 11:17:48

Hello all,
        A _huge_ scan was performed on our network this weekend from
129.62.1.75 (graduate-etd.baylor.edu) looking for anything with sunrpc op
Has anyone noticed anything similar from this IP address before?

462816 28Oct2000 11:11:11 AM drop tcp 129.62.1.75 10101 xxx.xxx.xxx.xxx
sunrpc

Thanks,
Abe
```

Compared to my scans above, we see this source IP was not one that hit us, but notice the source port, 10101. This mail was followed by James W. Abendschan who actually had a break-in,

```
Subject: Re: Network Scan - sunrpc
Date: Tue Oct 31 2000 11:47:56

On Mon, 30 Oct 2000, Abe Getchell wrote:
```

[Cut quote of previous mail. -cjc]

```
Yup.  Here's a slightly modified writeup I sent to CERT:

On September 26th, 2000, two of the RH 6.2 Linux machines where I work
were broken into.  The intruder apparently exploited a known bug in rpc.s
(unpatched on these two boxes, naturally.)

One of these boxes was being used to probe machines.  These probes
were sent to port 111/tcp and had a local port # of
10101/tcp.  Interestingly, these probes were used to determine the
remote host OS, not enumerate RPC services.

The initial overflow attack came around 13:02 PDT from from 216.253.64.10
As near as I can reconstruct, the sequence of events was as follows:
```

[Edited for length. Click here for the full archived message. I believe the rpc.statd vulnerability that Mr. Getchell refers to is likely CVE-2000-0666 (cool number, huh?).-cjc]

```
All this looks like the result of an automated script; if the timestamps
on the files are to be believed, it was installed after the statd exploit
in under 30 seconds.
```

[Snipped again. -cjc]

```
Shortly afterwards, I noticed the scanning activity from the box.
At the same time, complaints started coming in and I took the box
offline.  Two things struck me as interesting about this:

  1) They were scanning for host OS's only, not specific services.
  2) the ssh_host_key might suggest that the intruder owns (literally
     or figuratively) rapier.aerohead.com.
```

Finally, Guillaume Filion followed up with another mail claiming he had been hit by yet another source IP. What was interesting about his mail was that he got what looks like an IPchains log entry with more info about the packets,

```
Nov  3 22:26:52 cesam kernel: Packet log: input REJECT eth1 PROTO=6 24.200.yy.
```

We see that these are "real" SYN packets, no attempt to even be stealthy. Unfortunately, he did not post more than one to verify if the IP ID, 61434 for the packet above, is also static.

Going to the GIAC webpage to see what has been reported there, we find several more reports of scans to portmapper from source port 10101. There were a number of summaries with reports of these scans,

- Detects Analyzed 10/16/00
- Detects Analyzed 10/17/00
- Detects Analyzed 10/24/00
- Detects Analyzed 10/26/00
- Detects Analyzed 11/2/00
- Detects Analyzed 11/7/00
- Detects Analyzed 11/8/00

But nothing quite as interesting as the INCIDENTS exchange. However, some more complete dumps of packets showed that IP IDs changes, but the sequence number and acknowledgment number do not (100 and 0, respectively).

In the last entry, "Laurie@.edu," one of the regular posters to the GIAC page, did report getting hit by the same cable modem scanner, 24.20.135.187, on November 8th. In the same report, she also saw source port 10101 attacks being directed to 27374, the only reported scans not directed at port 111.

The timing is also very peculiar. Lots of activity for a few weeks and then none. Maybe this is not a widely distributed kiddie tool; those never seem to die. This could mean that we have a few people (the author(s) of the tool *might* be more than just skr1pt k1ddyz) using the tool from lots of different captured boxes.

Evidence of Active Targeting

The first scan walked across four full Class C nets, incrementing by one IP number, in just a few seconds. Minutes later, it banged across another Class C separated from the first four by ninety-three other C's. The other scans crossed a small 29-bit net we have routed to the firewall. This address space separated from those in the previous scan in the first octet of the IP.

The point is, these were loud, broad scans. They were scanning the whole Internet for vulnerable boxes. We were not actively targeted.

Severity

Target Criticality:	2	The one confirmed target was a Red Hat 6.2 box. We do not have any of those around, or any Linux-type operating systems at all. However, that is not proof that this was only going after exploits on those OSes. We have other UNIX machines that use RPC services that may be targets, so I am being conservative and giving this a 2.
Lethality:	5	This is an automated root exploit that someone reported haven fallen victim to. 'Nuff said.
System Countermeasures:	- 3	We try to keep the boxes patched, but many still run unnecessary services. Plus, we have no evidence that our OSes are being targeted.
Network Countermeasures:	- 5	This is exactly the kind of thing a firewall is perfect for blocking, random, automated, connect scans. There was essentially no risk of this getting past the firewall without going through the rules, and no rules allow any traffic like this through.
Severity:	- 1	A negative value tells us there is essentially no risk. It is a lethal attack, but exactly the kind of thing we have a firewall to block. We may have machines potentially vulnerable to RPC exploits, but none to the one exploit the tool has been confirmed to use.

Defensive Recommendation

Again, this is exactly the kind of thing a firewall is there to block. However, one should still try to keep up with patching the seemingly endless list of RPC program exploits. If not to protect yourself from the kiddies outside the firewall, then from the inside job.

Multiple Choice Question

The scans above are almost certainly the product of an automated, packet-crafting tool because,

a. The IP addresses are walked in numerical order.

b. All of the source port addresses are identical.
c. All of the destination port addresses are identical.
d. The packets are coming in very rapidly.

ANSWER: b. Nearly all operating system will change the source port, typically incrementing it by +1, each time a connection is initiated using the kernel's TCP/IP stack. Choices (a) and (c) would also be seen if the attacker was using a simple script. The last choice, (d), might be a indication that it is not a script, since they often are a bit slower, but a custom program using the kernel's stack would be just as fast, if not faster, than a packet crafting program.

Return to top.

Network Detect #4: Email Scanner

```
6Nov2000 11:05:27   drop >hme0  tcp 198.80.92.3:3717 -> AAA.BBB.31.192:mail 4
6Nov2000 11:05:32   drop >hme0  tcp 198.80.92.3:3718 -> AAA.BBB.31.193:mail 4
6Nov2000 11:05:37   drop >hme0  tcp 198.80.92.3:3721 -> AAA.BBB.31.195:mail 4
6Nov2000 11:05:42   drop >hme0  tcp 198.80.92.3:3722 -> AAA.BBB.31.196:mail 4
6Nov2000 11:05:47   drop >hme0  tcp 198.80.92.3:3723 -> AAA.BBB.31.197:mail 4
6Nov2000 11:05:52   drop >hme0  tcp 198.80.92.3:3724 -> AAA.BBB.31.198:mail 4
6Nov2000 11:05:57   drop >hme0  tcp 198.80.92.3:3725 -> AAA.BBB.31.199:mail 4
```

Source of Trace

Our company's Internet firewall which protects several class C networks.

Detect Generated by

Custom AWK scripts applied to "exported" Checkpoint Firewall-1 logs. See the description in Detect #1 for details.

Probability Source Address Spoofed

Unlikely. This looks like a connect scan and as we will see, a connection was actually made. This means the TCP three-way handshake was completed.

Description of Attack

We are scanned for SMTP (port 25) servers. Most SMTP scans are either spammers looking for open relays or kiddies looking to break into SMTP daemons with canned exploits. We will see this is most probably a kiddie looking for a box to steal.

Attack Mechanism

Our scanner is walking across the net one IP address at a time. From the rather slow pace of the

walk, five seconds spacing between connection attempts, it looks like he might not be using a specialized scanning tool. The source port is also in a reasonable range and stepping up once per connection attempt. This also points to a "real" connection attempt using the normal operating system.

But you have probably already noticed there is a hole in the scan at `AAA.BBB.31.194`. That is because there actually is an SMTP listener at that address. You also may notice that there are no logs of a successful connection included above. This is not because I filtered it out when processing the log. The log data shown above are all of the entries with that IP address, `198.80.92.3`, in any field. (So if my other three detects have not made the point already, firewall logs often leave much to be desired.)

The firewall, Checkpoint FW-1, has an SMTP proxy server which screens the mail for the real SMTP server, `AAA.BBB.31.194`. Messages that are received by the proxy are logged. Another log entry is entered when the message is either passed to the real SMTP server or when the mail is rejected. As we see above, there is no such entry.

After several tests with FW-1, I have determined there are two ways this can happen. If the sender connects to the proxy and gracefully terminates the session at the application level without trying to send a message of any kind,

```
$ telnet AAA.BBB.31.194 25
Trying AAA.BBB.31.194...
Connected to mail01.XXXXXX.com.
Escape character is '^]'.
220 CheckPoint FireWall-1 secure SMTP server
QUIT
221 Closing connection
Connection closed by foreign host.
```

Nothing is logged.

If the user interrupts the session at the application layer, but gracefully closes the TCP connection, as happens if we kill the telnet session,

```
$ telnet AAA.BBB.31.194 25
Trying AAA.BBB.31.194...
Connected to mail01.XXXXXX.com.
Escape character is '^]'.
220 CheckPoint FireWall-1 secure SMTP server
^]
telnet> q
Connection closed.
```

Nothing is entered in the mail FW-1 log, but we do get a log entry in the SMTP-proxy's own, separate log file,

```
18:08:53 fd: 7 src: 64.XXX.XXX.149 dst: AAA.BBB.31.194   Connection abort
```

We did not get such an entry when the session was terminated at the application layer.

Now, if we search the proxy's logs for our SMTP scanner,

```
11:05:37 fd: 8 src: 198.80.92.3 dst: AAA.BBB.31.194   Connection aborted
```

We find that at a time that meshes perfectly with our log entries above, he most unpolitely interrupted his session with the mail proxy.

From these facts, we can conclude he was most likely connecting to a STMP listener, examining the SMTP server's banner, and if it was not what he wanted to see, he immediately cut the connection. He was probably searching for versions of Sendmail, MS Exchange, *etc.* for which he has canned exploits.

Our scanner was probably not a spammer. In general, spammers will actually attempt to forward mail through the server. It is possible that we had a spammer who checked the banner first to try to guess if it was even worth the attempt. That, however, seems to greatly over estimate the IQ of your typical spammer; one of the few forms of 'Net life even lower than the script kiddies.

Correlations

Finding information on this particular source IP turned out to be difficult. The closest I could get doing a reverse-lookup of the address was,

```
;;      92.80.198.in-addr.arpa, type = ANY, class = IN

;; ANSWER SECTION:
92.80.198.in-addr.arpa.  22h47m33s IN NS  pic.net.
92.80.198.in-addr.arpa.  22h47m33s IN NS  ns.stb.com.

;; AUTHORITY SECTION:
92.80.198.in-addr.arpa.  22h47m33s IN NS  pic.net.
92.80.198.in-addr.arpa.  22h47m33s IN NS  ns.stb.com.
```

ARIN (whois.arin.net) verified that the address is associated with pic.net and NETCOM,

```
Registrant:
NETCOM Interactive (PIC2-DOM)
   1607 LBJ Freeway
   Dallas, TX 75234
   US

   Domain Name: PIC.NET
```

Searches of the GIAC site and the INCIDENTS@securityfocus.com archive provided no leads. Neither this address nor any others from its network (`198.80.92.3/24`) were present in our logs during this time frame. No other anomalous email activity from any addresses was noticed at the time of the scan.

Evidence of Active Targeting

This looks like a broad scan. The mail server they did find is pointed to by MX records for

several of our domains. If they wanted to target that machine, public information was available, but the fact the scan walked all of the way across the subnet makes that seem unlikely.

Severity

Target Criticality:	5	The target was our primary mail server.
Lethality:	2	From the evidence we have, it the attacker was just collecting banners and no evidence of overt hostile activity exists.
System Countermeasures:	- 4	In actuality, the system the scanner dealt with was the mail proxy. It is specifically designed to deal with attacks. It loses a point for the hole in the logging. It would be much more reassuring to know exactly what our scanner did rather than have to reconstruct it.
Network Countermeasures:	- 5	The firewall/proxy is there to handle these things. We already knocked off a point for the poor logging.
Severity:	- 2	There was essentially no risk our scanner could hurt our mail proxy. If he was a spammer, we are configured to prevent that as well.

Defensive Recommendation

If we want to receive email, and email is still the primary business use of the Internet, we need to have machines listening for incoming mail connections. People scanning for listening SMTP agents will bump into them. We cannot hide our SMTP servers, we need public MX records for email to work.

The scanners are almost certainly looking for unlisted, misconfigured, and forgotten SMTP agents for exploit or spamming. The main thing for security staff and system administrators to remember is to turn off SMTP listeners on systems that do not need them. Most UNIX-type systems, Solaris, IRIX, HP-UX, most Linux distributions, and the *BSD's (except OpenBSD), come with sendmail turned on out-of-the-box. Unless the machine needs it, and most do not, turn it off! And if it does need it, make sure to configure it correctly.

Multiple Choice Question

Which of the choices below is either not a practical option for stopping email spammers or is not an effective defense.

a. Turning off email services on machines that do not need to receive mail.
b. Keep your mail server software up to date and watch for vulnerability reports.
c. Properly configure email daemons to limit accepted relays to trusted sources.
d. Do not publically advertise your email server with DNS MX records.
e. Feed email spammers to wild dogs.

ANSWER: d. Yes, I am having a little fun here. Choice (e) is a gag, not that we wouldn't actually like to do it, but it is probably not practical, at least not in the US. Choice (d) is the real answer. It is not practical since MX records are essential for email to function on the Internet. People have to some how be able to figure out where to send mail

to your domain. All of the other choices, (a)-(c) are things that should be done to protect your systems from being abused by spammers.

Return to top.

Attack Evaluation: IDSwakeup

Introduction

While I was at the SANS 2000 Conference in Monterey, I happened to meet Stéphane Aubert from HSC at one of the birds of a feather (BoF) meetings. He had earlier in the week managed to defeat the IDSnet setup by several IDS vendors. Attendees were encouraged to try to break into a number of boxes as the vendors' IDSes watched. As part of his successful attempt (the only successful attempt of the week), he had used a tool of his creation called `IDSwakeup`. It is designed to test IDSes by throwing false attacks and anomalous traffic at them to see if they are triggered. The purpose is to check if the IDS falls over or false alarms occur at such a high rate as to overwhelm operators. And as Stéphane showed at the IDSnet, it is also a good way for an attacker to conceal the real attack. I have been meaning to take a look at this piece of software and this assignment seemed like the perfect excuse.

Building the tools

I downloaded the code and easily built the tool following the instructions provided. `IDSwakeup` requires a powerful packet crafting tool called hping2 (which has been analyzed in more than one previous practical) and the packet building library, libnet. Fortunately for me, I already had both of these installed on my systems. libnet has been ported to all the usual UNIX-type flavors and has a port to WindowsNT. However, hping2 is only reported to be fully supported on Linux and the *BSDs with a Solaris port in the works, no mention of a Windows port. (The following testing was done mainly on FreeBSD and OpenBSD platforms.)

Reading the Documentation and Examining the Code

There is not extensive documentation of the tool (perhaps this analysis can help to serve such a role for future users). However, the main routine, the `IDSwakeup` executable, is a well organized shell script that is readily accessible to the user. It creates traffic to mimic the following denial-of-service attacks, exploits, and suspicious activity in the order shown,

Teardrop	CAN-1999-0015, CAN-1999-0104, CAN-1999-0257, CAN-1999-0258 A potential DOS caused by insufficient sanity checks in IP fragment reassembly code. One of a family of such exploits. These were examined more closely below.
Land Attack	CVE-1999-0016 A potential DOS caused by insufficient sanity checks in the TCP/IP stack.
GET phf	CVE-1999-0067

	A CGI program that allows execution of commands on the server through shell metacharacters.
BIND Version	CVE-1999-0009, CVE-1999-0010 , CVE-1999-0011 , CVE-1999-0024 , CVE-1999-0184 , CVE-1999-0385 , CVE-1999-0833 , CVE-1999-0835 , CVE-1999-0837 , CVE-1999-0848 , CVE-1999-0849 , CVE-1999-0851 Queries for the BIND version are actually not an exploit for any of these vulnerabilities. Rather it is a frequent a precursor to attack. The attacker uses the query to determine which exploit to try.
GET phf SYN-ACK	CVE-1999-0067 A more sophisticated fake of a GET phf attack.
Ping of Death	CVE-1999-0128 A DOS caused by flaws in the IP fragment reassembly code. So named for exploits using oversized ICMP pings, but can be generalized to any IP protocol.
Syndrop	Another Teardrop variant.
Newtear	Another Teardrop variant. Named after the tool, newtear.c.
X11	Communications on the first XWindow System port (6000).
SMBnegprot	CVE-2000-0377 I believe this is related to a WinLogon DoS resulting from a malformed network request.
SMTP expr root	Watching for someone collecting reconnaissance on our systems.
finger Redirect	CAN-1999-0106 Some finger daemons allow redirection. This could be used for finger (port 79) bombing.
FTP cwd root	Appears like an attempt to change to the root users home directory during FTP session. (In the actual code, this attack is mislabeled as the "FTP PORT" signature.)
FTP PORT	CVE-1999-0017, Looks like an FTP bounce attempt. (In the actual code, this attack is mislabeled as the "FTP cwd root" signature.)
Trin00 Pong	CAN-2000-0138 Uses ports and passes traffic with strings associated with the Trin00 DDoS tool.
Back Orifice	Generates traffic on ports used by the popular remote system control package.
MSADCS	A Microsoft Internet Information Server (IIS) vulnerability.
WWW Fragments	Tiny fragments of an HTTP request.
WWW, Best of	CVE-1999-0148, CVE-1999-0191 A long list of other HTTP no-nos.
WWW, All	An even longer list of HTTP no-nos.
DDoS, Best of	Strings and port numbers that mimic DDoS signatures and server-zombie communications.

FTP, Best of	A number of ftp control channel signatures that may set off IDSs, e.g. suspicious PORT commands.
Telnet, Best of	A list of telnet abuses, e.g. username and password "cisco," "su root," dangerous environmental variables being passed, *etc.*
rlogin, Best of	A subset of the telnet abuses but aimed at the rlogin port (513).
TCP flags, Best of	Invalid TCP flag combinations usually associated with "stealth" scan attempts.
ICMP, Best of	Tries all valid ICMP codes. Passes data in ping packets.
SMTP, Bestof	Dangerous commands in SMTP sessions. Associated with a number of vulnerabilities on different mail daemons.
Misc, Best of	Stuff that did not fit in other categories but was not special enough for its own group, e.g. SNMP public strings, Napster-like profiles, *etc.*
DoS chargen	CVE-1999-0103 A UDP bomb between chargen (port 19) and echo (port 7).
DoS Snork	CVE-1999-0969 A DoS associated with WinNT RPC services.
DoS syslog	Sends a packet to the syslog (port 514, UDP) service with a non-syslog source port. This attack is associated with one of many syslog vulnerabilities on different platforms.

The "Best of" series are collections of multiple signatures that may set off IDSs. They are too numerous to go into here and most can be easily understood by examining the shell code.

Test Setup and Command Usage

`IDSwakeup` was run on an isolated network. The machine running `IDSwakeup` also recorded all traffic using `tcpdump`. A second machine running Snort listened as well. The attack was focused on a third machine. The "snort-rules" file distributed with Snort-1.6.3 was used to monitor the attacks.

For the tests, `IDSwakeup` was run with the following arguments,

```
$ IDSwakeup 10.10.10.10 192.168.64.254
```

The 10.10.10.10 address is a fake source to be put on crafted packets. Since I was going to listen on an actual network segment, at least the first hop to the destination needed to be valid. I chose the target to be the gateway address on the 192.168.64.0/24 network. This gateway host has a filter on its firewall rules to only allow in packets with a valid, 192.168.64.0/24, source. Since it is dropping all of these crafted packets at its firewall, I felt it was the safest host to attack.

`tcpdump` was run to capture all of the attack traffic,

```
$ tcpdump -w idswakeup0.dmp -s 1500
```

The arguments cause `tcpdump` to capture full packets (1500 is the MTU on the network segment) and write the data to a raw file for later analysis.

Finally, Snort was run with the following command,

```
$ snort -h 192.168.64.0/24 -A fast -c snort-lib -l IDSwakeup -D
```

These arguments tells Snort that (1) 192.168.64.0/24 is our home network, (2) it should write "fast" (one-line) alert messages, (3) it should use "snort-lib" as the rule file, (4) "IDSwakeup" is the directory to put the logs in, and (5) that it should run in daemon-mode. In addition, the "-r" option was used several times to run the `tcpdump` capture file through the Snort rules over again.

`tcpdump` Traces

In addition to using the hping2 tool, a simple command line packet crafter, `iwu`, came with the package. It has the very simple command line syntax of,

```
$ iwu srcIP dstIP nb ttl ip-datagram
```

Where *`nb`* is the number of packets to send and *`ttl`* it the IP time-to-live.

So, for example, the first attack, Teardrop, is written in the main `IDSwakeup` script as,

```
teardrop () {
  $IWU $src $dst $nb $ttl "\
                        4500 0038 00f2 2000 4011 53be 0101 0101 \
                        0202 0202 e63e 4494 0024 0000 0000 0000 \
                        0000 0000 0000 0000 0000 0000 0000 0000 \
                        0000 0000 0000 0000"
  $IWU $src $dst $nb $ttl "\
                        4500 0018 00f2 0003 4011 73db 0101 0101 \
                        0202 0202 e63e 4494"
}
```

Which makes our jobs as analysts almost too easy. The `tcpdump` capture of the packets generated by this code was,

```
$ tcpdump -nxvv -c 2 -r idswakeup1.dmp
23:13:25.698106 10.10.10.10.58942 > 192.168.64.254.17556:  udp 28 (frag 2
                        4500 0038 00f2 2000 2011 6409 0a0a 0a0a
                        c0a8 40fe e63e 4494 0024 0000 0000 0000
                        0000 0001 0406 0900 0000 0c0e 0000 0000
                        0000 0000 0000 0000
23:13:25.712861 10.10.10.10 > 192.168.64.254: (frag 548:4@14792) (ttl 32)
                        4500 0018 0224 0739 2011 7bbe 0a0a 0a0a
                        c0a8 40fe e63e 4494
```

We can see the changes made to the TTL, checksum, source, and destination fields. There also seems to be a problem in the TCP data field. I have not been able to determine where those values are coming from.

Several of the other attacks are also written in this `tcpdump`-like format. The `IDSwakeup` source code contains what amounts to a `tcpdump`-formatted list of interesting and common attacks. This is now combined this with the fact that I have gone through the research for the names and CVE numbers for as many as possible. I now have a nice, one-stop reference source for basic information on how these attacks actually look on the wire as well as the capability to generate them.

The other tool used to craft packets was hping2. However, since it has been covered multiple times in previous practicals, I will not discuss it in detail here.

Snort Traces

`IDSwakeup` definitely did cause Snort to alert heavily. These are the alerts generated by the default Snort ruleset,

```
11/11-02:00:39.108197  [**] Tiny Fragments - Possible Hostile Activity [*
11/11-02:00:39.122403  [**] Mostly Empty Fragmented Packet Discarded! [**
11/11-02:00:45.178020  [**] Mostly Empty Fragmented Packet Discarded! [**
11/11-02:00:48.207928  [**] MISC-DNS-version-query [**] 10.10.10.10:1249
11/11-02:00:54.277707  [**] Tiny Fragments - Possible Hostile Activity [*
11/11-02:00:57.308657  [**] Tiny Fragments - Possible Hostile Activity [*
11/11-02:01:21.568308  [**] Back Orifice [**] 10.10.10.10:1249 -> 192.168
11/11-02:01:24.648513  [**] IIS-msadc/msadcs.dll [**] 10.10.10.10:1549 ->
11/11-02:01:37.730244  [**] Tiny Fragments - Possible Hostile Activity [*
11/11-02:01:37.731084  [**] WEB-/.... [**] 10.10.10.10:2198 -> 192.168.64
11/11-02:01:47.808706  [**] Tiny Fragments - Possible Hostile Activity [*
11/11-02:02:10.964258  [**] IDS227 - Web-CGI-Scriptalias [**] 10.10.10.10
11/11-02:02:31.103831  [**] SCAN-Whisker! [**] 10.10.10.10:1284 -> 192.16
11/11-02:02:51.245439  [**] WEB-CGI-Webdist CGI access attempt [**] 10.10
11/11-02:03:01.314801  [**] SCAN - Whisker Stealth- mlog access attempt [
11/11-02:03:11.384480  [**] SCAN - Whisker Stealth- mylog access attempt
11/11-02:04:01.739859  [**] WEB-CGI-Wrap CGI access attempt [**] 10.10.10
11/11-02:04:11.811353  [**] IDS128 - CVE-1999-0067 - CGI phf attempt [**]
11/11-02:05:02.162042  [**] CVE-1999-0278 - IIS-asp [**] 10.10.10.10:2383
11/11-02:05:52.513005  [**] FrontPage-administrators.pwd [**] 10.10.10.10
11/11-02:06:02.582323  [**] IDS235 - CVE-1999-0148 - CGI-HANDLERprobe! [*
11/11-02:06:12.652695  [**] WEB-etc/passwd [**] 10.10.10.10:2205 -> 192.1
11/11-02:06:22.723679  [**] IDS219 - WEB-CGI-Perl access attempt [**] 10.
11/11-02:06:32.793689  [**] CVE-1999-0191 - IIS-newdsn [**] 10.10.10.10:2
11/11-02:06:42.863800  [**] IIS-search97 [**] 10.10.10.10:2835 -> 192.168
11/11-02:07:06.038009  [**] IDS111 - DDoS - mstream client to handler [**
11/11-02:07:16.123703  [**] IDS110 - DDoS - mstream handler to client [**
11/11-02:07:26.216487  [**] IDS110 - DDoS - mstream client to handler [**
11/11-02:07:36.308789  [**] IDS103 - DDoS - mstream agent pong to handler
11/11-02:07:46.398601  [**] IDS102 - DDoS - mstream handler ping to agent
11/11-02:07:56.489524  [**] IDS101- DDoS - mstream handler to agent [**]
11/11-02:08:06.579785  [**] IDS100 - DDoS - mstream agent to handler [**]
11/11-02:08:16.648727  [**] DDoS - Trin00 Attacker to Master-default mdie
11/11-02:08:26.739814  [**] IDS187 - DDoS - Trin00 [**] 10.10.10.10:1714
11/11-02:08:56.989784  [**] DDoS - Trin00 Attacker to Master defaultr.i.p
11/11-02:09:07.059389  [**] IDS254 - DDoS shaft client to handler [**] 10
11/11-02:09:17.151331  [**] IDS255 - DDoS shaft handler to agent [**] 10.
11/11-02:09:27.241721  [**] IDS256 - DDoS shaft agent to handler [**] 10.
11/11-02:09:37.331667  [**] IDS252 - DDoS shaft synflood incoming [**] 10
```

```
11/11-02:10:20.620830  [**] IDS213 - FTP-Password Retrieval [**] 10.10.10
11/11-02:10:30.691630  [**] FTP-site-exec [**] 10.10.10.10:2231 -> 192.16
11/11-02:10:40.762349  [**] FTP-site-exec [**] 10.10.10.10:2441 -> 192.16
11/11-02:10:50.832657  [**] IDS213 - FTP-Password Retrieval [**] 10.10.10
11/11-02:11:10.972319  [**] FTP-cwd~root [**] 10.10.10.10:1071 -> 192.168
11/11-02:12:54.689433  [**] TELNET - resolv_host_conf [**] 10.10.10.10:15
11/11-02:13:04.758924  [**] TELNET - ld_preload [**] 10.10.10.10:1769 ->
11/11-02:41:24.288797  [**] spp_portscan: PORTSCAN DETECTED from 10.10.10
11/11-02:14:21.295581  [**] SCAN-SYN FIN [**] 10.10.10.10:1716 -> 192.168
11/11-02:41:24.289799  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:41:24.290248  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:41:24.290992  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:41:24.291433  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:41:24.291831  [**] spp_portscan: End of portscan from 10.10.10.1
11/11-02:41:24.292684  [**] spp_portscan: PORTSCAN DETECTED from 10.10.10
11/11-02:41:24.293151  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:41:24.293592  [**] spp_portscan: portscan status from 10.10.10.1
11/11-02:16:05.325368  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:16:15.396019  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:16:25.465870  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:16:35.536359  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:16:45.606840  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:16:55.676392  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:17:05.746775  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:17:15.817273  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:17:25.886849  [**] ICMP Destination Unreachable [**] 10.10.10.10
11/11-02:17:35.957038  [**] ICMP Source Quench [**] 10.10.10.10 -> 192.16
11/11-02:18:26.308614  [**] IDS162 - PING Nmap2.36BETA [**] 10.10.10.10 -
11/11-02:18:36.378723  [**] ICMP Time Exceeded [**] 10.10.10.10 -> 192.16
11/11-02:18:46.448906  [**] ICMP Time Exceeded [**] 10.10.10.10 -> 192.16
11/11-02:19:10.039426  [**] IDS031 - SMTP-expn-root [**] 10.10.10.10:1615
11/11-02:19:20.109852  [**] IDS032 - SMTP-expn-decode [**] 10.10.10.10:18
11/11-02:41:24.305869  [**] spp_portscan: End of portscan from 10.10.10.1
11/11-02:20:43.586775  [**] SNMP public access [**] 10.10.10.10:1610 -> 1
11/11-02:21:23.926776  [**] MISC-PCAnywhere Attempted Administrator Login
11/11-02:21:34.017267  [**] MISC-Attempted Sun RPC high port access [**]
11/11-02:21:44.086770  [**] Napster Client Data [**] 10.10.10.10:1153 ->
11/11-02:21:54.156574  [**] Napster 8888 Data [**] 10.10.10.10:1363 -> 19
11/11-02:22:04.226758  [**] Napster 7777 Data [**] 10.10.10.10:1573 -> 19
11/11-02:22:14.296570  [**] Napster 6666 Data [**] 10.10.10.10:1783 -> 19
11/11-02:22:24.366888  [**] Napster 5555 Data [**] 10.10.10.10:2345 -> 19
11/11-02:22:34.436645  [**] Napster 4444 Data [**] 10.10.10.10:2555 -> 19
11/11-02:22:44.507419  [**] Napster Server Login [**] 10.10.10.10:2765 ->
```

If you note the timestamps on the alerts, you will notice that they are coming in fairly slowly. In the wild, `IDSwakeup` can run at a very much faster pace. I altered the source code to slow down the attack for a number of reasons during my testing. It would be quite easy to generate all of the above alerts in a second or two.

The Snort scan report contains information about the nonsense TCP flag combinations sent. It looks something like a fingerprint attempt.

```
Nov 11 02:14:21 10.10.10.10:1716 -> 192.168.64.254:80 SYNFIN **SF****
Nov 11 02:14:31 10.10.10.10:1609 -> 192.168.64.254:80 NOACK **S*R***
Nov 11 02:14:41 10.10.10.10:1706 -> 192.168.64.254:80 NULL ********
```

```
Nov 11 02:15:01 10.10.10.10:1688 -> 192.168.64.254:80 NOACK **SFR***
Nov 11 02:15:11 10.10.10.10:1629 -> 192.168.64.254:80 INVALIDACK 21SFRPA*
Nov 11 02:15:31 10.10.10.10:1747 -> 192.168.64.254:80 INVALIDACK **SFR*A*
Nov 11 02:15:42 10.10.10.10:1668 -> 192.168.64.254:80 NULL 21****** RESER
Nov 11 02:15:52 10.10.10.10:1761 -> 192.168.64.254:1999 SYN **S*****
```

`IDSwakeup` Signatures

While examining the `IDSwakeup` script, it is almost immediately obvious that `IDSwakeup` has some recognizable signatures of its own. (Although it does seems somewhat ironic to be considering building signatures for a tool that is intended to create false signatures.) Besides the the obvious feature that it would trigger ridiculously many alarms in a set sequence on most any IDS, there are multiple fields in the crafted packets that will be identical each time. For example, in the Teardrop attack examined earlier, the source and destination ports will always be 58942 and 17556, respectively. One could conceivably compile a list of all such quirks in the code. However, it is trivial for the user to change the values in the script, even using the pseudo-random number generator already built into the script making the values change each run.

Still, we all are aware of how rarely attackers actually have either the skills or motivation to modify attack tools. The best identifying feature for an unmodified `IDSwakeup` attack will be the order of the alerts. They will generally occur in the order shown in the Snort alert output above. However, if the attack is being run from a remote location, there can be some slight jostling of the order if the packets do not arrive in the order they were sent.

This analysis also seems to have found a `Snort` bug. I found `IDSwakeup` can quite reliably seg-fault Snort 1.6.3 while I was using another ruleset (I was originally planning to compare the results of the two sets). I reproduced the problem on both FreeBSD and OpenBSD platforms. Once I narrow it down a bit more, verify it is a real bug and not a problem with my build, I will be in contact with Marty. Any problems or solutions will be posted at BugTraq.

Return to top.

"Analyze This"

Introduction

As part of a bidding process, we were asked to review a several weeks of network logs for intrusion detection purposes. The data was gathered by Snort-based intrusion detection systems on the network of GIAC Enterprises. Several megabytes of collection data was made available to us. However, we were provided no information about the network architecture, current security procedures, practices, or policies. We did have access to earlier analyses of this network by groups doing similar project. But we did not have access to the actual data they used.

The vast majority of the patterns in the network logs have been described to you multiple times in the previous reports. Below, we try to avoid going into depth on topics that have already been brought to your attention.

The Computer Network Center Chinese Academy of Sciences

Your network is subject to heavy probing from the 159.226.0.0/16 netblock which belongs to,

```
The Computer Network Center Chinese Academy of Sciences (NET-NCFC)
   P.O. Box 2704-10,
   Institute of Computing Technology Chinese Academy of Sciences
   Beijing 100080, China

   Netname: NCFC
   Netnumber: 159.226.0.0

   Coordinator:
      Qian, Haulin  (QH3-ARIN)  hlqian@NS.CNC.AC.CN
      +86 1 2569960

   Domain System inverse mapping provided by:

   NS.CNC.AC.CN                159.226.1.1
   GINGKO.ICT.AC.CN            159.226.40.1

   Record last updated on 25-Jul-1994.
   Database last updated on 20-Nov-2000 06:10:16 EDT.
```

From your logs, it does appear that your network and security administrators are well aware of this fact. Reviews of the previous security evaluations of your network show that the NCFC network has historically shown this behavior.

Our analysis as well as previous ones noted significant STMP (email, port 25) traffic between the networks. With just the information available it is difficult to determine whether this was "legitimate" email traffic or not. We can determine that the initial TCP connections were completed and there was likely some exchange of data. Whether a message was actually sent is difficult to determine. Here is a typical trace from the host 159.226.63.200,

```
08/11-02:44:27.163055  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:28.112227  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:29.082632  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:29.960565  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:29.982611  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.028586  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.028637  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.583344  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.979301  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.989213  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:30.991029  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:31.966763  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:31.966876  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:44:32.595327  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:45:54.286460  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:45:58.285486  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:45:58.393495  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:45:58.393540  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:45:58.399331  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:46:00.574367  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
```

```
08/11-02:46:01.397970  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:46:04.270275  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:27.880074  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:27.880711  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:28.709646  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:28.709694  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:28.709742  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:28.717819  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:30.557600  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:30.721803  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:31.459016  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:31.531969  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:31.532012  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:32.416612  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:32.416729  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:56.144008  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:56.144227  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:47:56.718868  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
08/11-02:49:21.030334  [**] Watchlist 000222 NET-NCFC [**] 159.226.63.200
```

We see what looks like three separate connections made to port 25 over a three minute time span. Each time, the email server in MY.NET responded with a ident query (port 113) and we see the response coming back. It looks like there is an initial exchange, a pause (perhaps a timeout), before some final activity (possibly closing up the connection).

The machines MY.NET.253.41, MY.NET.253.42, and MY.NET.253.43 have had several thousand SMTP (email, port 25) connection attempts during the period examined. From the data it appears these machines do have email services enabled. Below is the list of hosts from the NCFC netblock which have made SMTP connection attempts and the number of attempts,

IP Address	No. of Attempts	IP Address	No. of Attempts
159.226.63.190	9798	159.226.64.61	10
159.226.45.3	1209	159.226.8.6	9
159.226.63.200	988	159.226.39.1	6
159.226.5.77	254	159.226.156.1	6
159.226.21.3	144	159.226.118.1	6
159.226.5.222	114	159.226.64.138	5
159.226.115.1	87	159.226.116.129	5
159.226.64.152	58	159.226.114.1	5
159.226.23.155	57	159.226.47.196	4
159.226.5.94	51	159.226.5.225	3
159.226.228.1	48	159.226.128.1	3
159.226.66.130	25	159.226.91.20	2
159.226.158.188	19	159.226.99.1	1
159.226.159.1	18	159.226.68.65	1

159.226.5.65	13
159.226.5.188	12
159.226.22.30	11

159.226.41.188	1
159.226.128.8	1

Below are the destinations of these connection attempts. Over 95% of the attempts were made to three machines on your network,

IP Address	No. of Attempts
MY.NET.253.43:25	4839
MY.NET.253.42:25	3842
MY.NET.253.41:25	3750
MY.NET.100.230:25	421
MY.NET.6.7:25	173
MY.NET.110.150:25	11
MY.NET.6.35:25	10
MY.NET.1.2:25	6
MY.NET.97.181:25	5
MY.NET.6.34:25	5
MY.NET.179.80:25	1
MY.NET.1.14:25	1

Without mail logs or information about what is passing in the TCP stream, it is difficult to diagnose this behavior further. Are the top three destinations listed above mail servers? These connection attempts generated a large number of ident (port 113) responses in the reverse direction. This indicates that there is indeed a SMTP listener on these machines. If these machines are servers, the SMTP server logs should be watched closely and the SMTP configuration properly maintained. If these are not mail servers, SMTP services should be deactivated.

isdn.net.il

The netblock 212.179.0.0/17,

```
domain:      isdn.net.il
descr:       Bezeq International
descr:       40  hashaham st Petach-Tikva
descr:       Israel
phone:       +972 3 9257790
fax-no:      +972 3 9257795
e-mail:      ori@bezeqint.net
admin-c:     OR214-IL
tech-c:      OR214-IL
zone-c:      NP469-IL
```

```
nserver:     ns1.bezeqint.net
nserver:     ns2.bezeqint.net
mnt-by:      NET-IL-DNS
changed:     registrar@ns.il 19971013 (Assigned)
changed:     registrar@ns.il 19971202 (Changed)
changed:     registrar@ns.il 19990629 (Changed)
changed:     registrar@ns.il 19990907 (Changed)
source:      IL
```

Showed continued interest in your networks and hosts. Again, this is a block which your IDS is already closely watching and has been the subject of multiple analyses and recommendations in previous reports. Specific incidents are dealt with below.

Public SNMP

A number of internal hosts continue to try to access host MY.NET.101.192 using default "public" SNMP settings. This has been repeatedly referenced in earlier reports. We can only assume that the administrators of the network do not perceive this to be a threat. We would note that no external (outside of MY.NET.0.0/16) SNMP traffic was monitored.

Examples of the alerts,

```
08/15-19:59:52.855020  [**] SNMP public access [**] MY.NET.97.154:1049 ->
08/15-20:00:04.771520  [**] SNMP public access [**] MY.NET.97.154:1050 ->
08/15-20:00:05.128465  [**] SNMP public access [**] MY.NET.97.154:1058 ->
08/15-20:00:05.347645  [**] SNMP public access [**] MY.NET.97.154:1059 ->
08/15-20:00:05.896381  [**] SNMP public access [**] MY.NET.97.154:1060 ->
```

And the list of machines that attempted to access SNMP on MY.NET.101.192,

MY.NET.97.154	MY.NET.97.206	MY.NET.97.217	MY.NET.97.244
MY.NET.97.246	MY.NET.98.109	MY.NET.98.114	MY.NET.98.148
MY.NET.98.159	MY.NET.98.171	MY.NET.98.172	MY.NET.98.177
MY.NET.98.181	MY.NET.98.190	MY.NET.98.191	MY.NET.98.201

Sun RPC Access Reports

A large number of alerts were generated for "Attempted Sun RPC high port access." Many of these were between port 4000 on the remote machine and 32771 on the local (on MY.NET) machine, e.g.,

```
09/02-11:28:18.700102  [**] Attempted Sun RPC high port access [**] 205.1
09/02-11:30:18.587762  [**] Attempted Sun RPC high port access [**] 205.1
09/02-11:31:18.724356  [**] Attempted Sun RPC high port access [**] 205.1
09/02-11:33:50.060535  [**] Attempted Sun RPC high port access [**] 205.1
09/02-11:55:17.743505  [**] Attempted Sun RPC high port access [**] 205.1
09/02-11:56:17.517864  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:04:19.150958  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:06:20.197634  [**] Attempted Sun RPC high port access [**] 205.1
```

```
09/02-12:07:22.893069  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:09:24.013527  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:13:25.015051  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:19:28.516129  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:38:37.579259  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:41:40.298888  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:42:40.785940  [**] Attempted Sun RPC high port access [**] 205.1
09/02-12:45:44.968047  [**] Attempted Sun RPC high port access [**] 205.1
```

The source address, 205.188.153.109, is listed as an ICQ server,

```
;; ANSWER SECTION:
109.153.188.205.in-addr.arpa.  1H IN PTR  fes-d013.icq.aol.com.
```

This could be usage of the ICQ service. But it is also possible that it is an attempt to circumvent a firewall to access RPC services (port 32771 is frequently where a "ghost" portmapper lives) and hide it as ICQ.

This is the same conclusion previous analysts have reached. The local hosts involved should be examined, the users of the machines asked if they have been using ICQ, and the continued use of ICQ should be studied with respect to your site's security policy.

Another group of such alerts,

```
09/06-23:10:10.012419  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:10.159763  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:10.302667  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:10.312668  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:11.455277  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:12.021403  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:12.021491  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:12.021997  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:12.173462  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:17.091000  [**] SUNRPC highport access! [**] 193.64.205.17:56
09/06-23:10:20.590610  [**] SUNRPC highport access! [**] 193.64.205.17:56
.
.
.
```

Occurs with a host registered to,

```
% Rights restricted by copyright. See http://www.ripe.net/ripencc/pub-ser

inetnum:     193.64.205.16 - 193.64.205.31
netname:     SSH-FI-2
descr:       SSH Communications Security Ltd.
descr:       Fredrikinkatu 42, 00100 Helsinki
country:     FI
admin-c:     JS19169-RIPE
tech-c:      JS19169-RIPE
status:      ASSIGNED PA
notify:      hostmaster@kpnqwest.fi
mnt-by:      KQFI-NOC-MNT
```

```
changed:     hostmaster@kpnqwest.fi 20001023
source:      RIPE
```

This may be high ports used for a FTP data connection. However, the local machine did show other odd activity on this port,

```
09/07-21:10:18.892811  [**] SUNRPC highport access! [**] 207.29.195.22:26
09/11-21:24:53.037663  [**] SUNRPC highport access! [**] 209.10.41.242:21
09/11-21:24:53.037663  [**] SUNRPC highport access! [**] 209.10.41.242:21
```

So it is possibly worth checking up on.

SOCKS/WinGate

Previous analyses (practicals) warned that the WinGate/SOCKS scans were signs of compromised machines. However, a quick survey of the machines scanning you for SOCKs shows a very large proportion are IRC servers. As previously referenced above, this is often a defensive action of IRC servers. The most appropriate action is to consider your corporate policy on allowing users to access IRC servers.

Out of the 192 machines attempting WinGate connections, 45 were appeared to be running IRC servers at the time of this writing.

IP Address	Host Name	IP Address	Host Name
142.177.93.59	hlfx43-59.ns.sympatico.ca	209.133.28.137	webmaster.ca.us.webchat
168.120.16.250	irc.au.ac.th	209.133.35.87	irc2.bondage.com
192.216.128.28	protege.linkline.com	209.133.35.89	irc3.bondage.com
194.159.247.49	draconic.fyremoon.net	209.235.102.9	www-virt-atl.dns-host.co
194.75.152.237	dalnet.lineone.net	212.29.92.167	
198.182.76.100	irc.blackened.com	212.72.0.73	
199.120.223.4	cosmos.lod.com	213.188.7.2	login1.ssc.net
199.172.149.141		216.152.64.129	sauron.ca.us.webchat.org
203.85.89.3	princess.os.st	216.152.64.130	glass.webmaster.com
204.228.135.144	poseidon.mediabang.com	216.152.64.137	glass2.webmaster.com
205.252.89.153	sana-chan.animenet.org	216.152.64.150	stable.ca.us.webchat.org
207.126.106.118	mandarin.peopleweb.com	216.152.64.151	katana.ca.us.webchat.org
207.151.147.201	pool.207.151.147.201.cinenet.net	216.152.64.213	w2k.webmaster.com
207.25.80.133		216.234.161.197	mircx.com
207.25.80.134		216.252.144.239	ipg239.compass.com.ph
207.7.26.15	hermes.acronet.net	216.35.116.103	wm3018.inktomi.com
208.178.165.228	pub.iad.redhat.com	216.35.116.90	si3000.inktomi.com

208.199.34.160	
208.5.1.212	unassigned.starnet.com.eg
209.1.224.129	oldftp4.geocities.com
209.1.224.160	oldftp1.geocities.com
209.1.233.136	
209.10.218.250	java.webmaster.com

216.35.116.91	si3001.inktomi.com
24.113.168.58	cr277399-a.nvcr1.bc.wave.home.cc
63.160.118.9	cols631601189.cols.net
63.238.214.65	irc.friendlynet.com
64.229.194.201	HSE-MTL-ppp70868.qc.sympatico.c

Happy 99 Virus

Snort detected the Happy 99 worm possibly being sent to two machines, MY.NET.6.35:25 on August 16th and MY.NET.179.80:25 on August 20th. The mail logs on those machines should be reviewed so the recipients can be found and any required action taken.

New Findings: Telnet

Among all of the traffic captured by the filters for NCFC there seems to be a new and somewhat alarming signature. There is extensive TELNET (port 23) traffic between MY.NET and NCFC. Host 159.226.45.108 appears to have connected with MY.NET.6.7 from 0151 to 0223 on August 11th. At that point it appears to pick up a TELNET with MY.NET.60.8 which is active until 0225. A new session with MY.NET.6.7 begins at this point and runs until 0301. Nothing is seen again until host 159.226.45.3 connects to MY.NET.6.7 from 2042 until 2050 on August 16th. No other, similar incoming TELNET signatures were found. The first and last trace entries for each is shown,

```
08/11-01:51:05.043450  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108
...
08/11-02:23:28.664773  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108

08/11-02:23:44.834384  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108
...
08/11-02:25:17.278243  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108

08/11-02:25:24.710017  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108
...
08/11-02:34:48.216685  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108

08/11-02:33:24.097140  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108
...
08/11-03:01:53.618330  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.108

08/16-20:42:09.047163  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.3:4
...
08/16-20:50:55.336179  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.3:4
```

There is what appears to be an *outgoing* telnet connection from MY.NET.98.168 to 212.179.58.2 which lies in the suspicious isdn.net.il block. The connection lasted from 1135 to 1151 on August 11th. Later that day, from 1611 to 1621 there appears to be an outgoing TELNET from

MY.NET.60.11 to 159.226.41.166 in the NCFC block. Finally, on September 11th from 1058 until 1117 there appears to be a TELNET from MY.NET.163.32 to 159.226.45.3.

```
08/11-11:35:46.194548  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
...
08/11-11:51:55.268612  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1

08/11-16:11:43.712817  [**] Watchlist 000222 NET-NCFC [**] 159.226.41.166
...
08/11-16:21:53.440338  [**] Watchlist 000222 NET-NCFC [**] 159.226.41.166

09/11-10:58:58.817818  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.3:2
...
09/11-11:17:22.505410  [**] Watchlist 000222 NET-NCFC [**] 159.226.45.3:2
```

These local hosts should be investigated for intrusions if they have not been already. MY.NET.6.7 seems to be a particular focus of NCFC TELNET and SMTP activity. It also was portscanned by 195.57.243.171 which belongs to a Spansish ISP,

```
% Rights restricted by copyright. See http://www.ripe.net/ripencc/pub-ser

inetnum:     195.57.243.0 - 195.57.243.255
netname:     BITELNET
descr:       Internet Service Provider
country:     ES
admin-c:     JAM1-RIPE
admin-c:     JB8545-RIPE
tech-c:      JB8545-RIPE
tech-c:      JAM1-RIPE
status:      ASSIGNED PA
mnt-by:      MAINT-AS3352
changed:     felicidad.martin@telefonica-data.com 19991115
changed:     felicidad.martin@telefonica-data.com 19991116
changed:     felicidad.martin@telefonica-data.com 20000621
source:      RIPE
```

Host MY.NET.53.28 may have been compromised. There was unusual traffic to sent by 192.117.131.201, and then days later we see a great deal of traffic to a suspicious host, 212.179.29.150.

```
Aug 16 15:45:12 192.117.131.201:1129 -> MY.NET.53.28:4407 NOACK *1SFRP**

08/20-09:05:41.982563  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:42.671792  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:44.418409  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:44.420038  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:44.856254  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:44.858776  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:05:44.900843  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
.
.
.
08/20-09:08:30.055172  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
08/20-09:08:30.366963  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
```

```
08/20-09:08:38.725691  [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.1
```

The integrity of host MY.NET.53.28 should be checked if it has not been already.

Napster, Again

As discussed in the "Napster Strikes Back" trace above, there appear to be a number of "bounce-back" connections from Napster machines with unusual properties. Some hosts that showed this behavior and the local machine that was attacked include,

Remote Host	Local Host
128.138.14.148	MY.NET.179.52
128.194.51.187	MY.NET.210.114
128.226.152.34	MY.NET.206.114
128.61.105.106	MY.NET.218.202
128.61.59.79	MY.NET.217.34
129.59.24.21	MY.NET.204.126
129.93.214.47	MY.NET.223.186
130.239.11.230	MY.NET.181.173
130.239.142.167	MY.NET.223.58
130.49.220.26	MY.NET.226.6
132.199.220.223	MY.NET.205.26
137.82.136.39	MY.NET.97.150
139.91.171.50	MY.NET.211.234
141.40.205.133	MY.NET.224.34
150.216.127.179	MY.NET.206.66
193.251.71.243	MY.NET.146.68
194.237.99.150	MY.NET.223.38
195.132.204.48	MY.NET.220.154

Remote Host	Local Host
200.145.151.163	MY.NET.221.114
200.52.201.4	MY.NET.217.222
207.230.248.254	MY.NET.208.18
212.33.70.83	MY.NET.206.66
216.63.200.250	MY.NET.203.106
24.112.241.246	MY.NET.201.58
24.160.189.151	MY.NET.220.206
24.164.181.31	MY.NET.217.26
24.180.196.93	MY.NET.217.250
24.19.101.91	MY.NET.222.46
24.22.125.94	MY.NET.223.38
24.232.79.188	MY.NET.206.114
24.28.33.193	MY.NET.224.134
24.29.7.199	MY.NET.218.94
24.8.241.127	MY.NET.70.217
24.92.174.232	MY.NET.217.130
24.92.188.4	MY.NET.106.164
62.2.64.86	MY.NET.218.10

A significant amount of the Napster-like traffic involved hosts on the isdn.net.il watchlist. The hosts pairings involved were,

Remote Host	Local Host
212.179.127.45	MY.NET.202.58
212.179.58.174	MY.NET.157.200
212.179.58.204	MY.NET.205.254
212.179.66.2	MY.NET.181.87
212.179.66.2	MY.NET.221.94

212.179.67.195	MY.NET.208.178

Interestingly, there is no overlap between the two groups.

The use of Napster-like applications by users should be considered with respect your security policy. If users should not be using it, block it where possible and investigate unauthorized use. If it is decided that file sharing will be tolerated, educate users on how to use the tools safely.

FTP

Right now, almost daily FTP scans are part of living on the Internet. Your network was no exception. The following hosts conducted FTP scans of your net,

IP Address	No. Hosts Scanned	IP Address	No. Hosts Scanned
195.114.226.41	42652	206.18.105.224	56
24.17.189.83	20155	195.130.128.202	42
212.141.100.97	19931	64.1.198.164	40
194.165.230.250	3300	24.180.134.156	31
210.61.144.125	2411	140.134.26.58	20
212.170.19.199	823	212.143.237.22	19
4.54.37.160	814	207.151.147.201	5
24.94.176.113	589	131.155.192.220	2
212.41.61.40	291	24.180.174.167	1
213.188.8.45	227	216.99.200.242	1
210.100.192.254	225	198.62.155.11	1

The scanner 24.17.189.83, listed as the number two culprit above, was detected actually attempting different WU-FTP exploits (CVE-1999-0080 and one not clearly identified by the logs) on,

MY.NET.99.104
MY.NET.150.24
MY.NET.202.190
MY.NET.202.202

Also, there is evidence that the scanning host 212.141.100.97 may have established data connections with the following hosts,

MY.NET.130.116
MY.NET.130.190

MY.NET.201.26
MY.NET.5.7

These hosts should be checked for compromise and to see what data the scanners may have had access to.

Port Scans

A number of machines were portscanned. In additions to searches for open ports (SYN scans), attempts to fingerprint the operating system were conducted where indicated.

Target	Source	Fingerprint
129.21.145.131	MY.NET.150.24	YES
131.155.192.220	MY.NET.5.7	YES
134.28.9.225	MY.NET.111.67	
141.30.227.175	MY.NET.223.106	YES
144.132.5.130	MY.NET.217.218	YES
147.208.171.139	MY.NET.150.89	
147.208.171.139	MY.NET.203.86	
147.208.171.139	MY.NET.97.230	
147.208.171.139	MY.NET.98.160	
151.196.73.119	MY.NET.253.112	YES
168.120.13.177	MY.NET.97.248	
168.120.26.87	MY.NET.97.248	
192.193.195.178	MY.NET.53.41	YES
195.57.243.171	MY.NET.6.7	
195.57.243.171	MY.NET.60.8	
198.62.155.10	MY.NET.217.10	
198.62.155.101	MY.NET.217.10	
198.62.155.102	MY.NET.217.10	
198.62.155.103	MY.NET.217.10	
198.62.155.104	MY.NET.217.10	
198.62.155.105	MY.NET.217.10	
198.62.155.106	MY.NET.217.10	
198.62.155.107	MY.NET.217.10	
198.62.155.109	MY.NET.217.10	
198.62.155.11	MY.NET.217.10	
198.62.155.111	MY.NET.217.10	

Target	Source	Fingerp
207.102.30.6	MY.NET.218.62	YES
207.123.169.54	MY.NET.202.150	
207.123.169.54	MY.NET.217.206	
207.123.169.54	MY.NET.220.190	
207.151.147.201	MY.NET.60.8	YES
207.236.3.96	MY.NET.20.10	
209.123.109.175	MY.NET.207.74	
209.123.109.175	MY.NET.219.118	
209.70.118.223	MY.NET.60.11	
210.125.174.11	MY.NET.97.199	
210.56.17.202	MY.NET.98.201	
212.242.100.15	MY.NET.218.130	YES
216.138.44.49	MY.NET.98.129	
216.234.161.76	MY.NET.218.34	
216.99.200.242	MY.NET.97.209	
216.99.200.242	MY.NET.97.216	
216.99.200.242	MY.NET.98.188	
24.112.134.95	MY.NET.218.70	YES
24.113.80.28	MY.NET.203.110	YES
24.12.112.239	MY.NET.205.26	
24.13.114.3	MY.NET.60.8	
24.180.174.167	MY.NET.253.42	
24.180.174.167	MY.NET.253.52	
24.180.174.167	MY.NET.60.11	
24.180.196.93	MY.NET.217.250	YES
24.23.198.174	MY.NET.217.46	YES

203.130.2.59	MY.NET.98.201		24.6.140.249	MY.NET.130.190	YE
203.176.29.200	MY.NET.97.144		38.179.244.220	MY.NET.60.11	
203.176.29.202	MY.NET.97.218		63.144.227.21	MY.NET.208.190	YE
203.176.29.202	MY.NET.98.164		63.226.208.41	MY.NET.253.41	YE

An honorable mention must go to 24.180.134.156 who did an NMap fingerprint and TCP connect scan of the entire MY.NET.208.0/24 subnet. His efforts were not included above for the sake of space.

Other than the several, possibly spoofed, hosts that scanned MY.NET.217.10, there seems to be little pattern to the targeting. We presently have no evidence that anything more than scans of the hosts were conducted in these examples.

Host Scans

Your network was subject to a number of broad host scans for the usual Windows trojans, BackOrifice, Subseven, Netbus, *etc.* Except for the FTP scans and WinGate/SOCKS issues referenced above, none of the scans appears to be a significant threat. Since these have been dealt with extensively in previous evaluations, we will forego further analysis of these scans.

NetBIOS

In August, there were a number of alerts dealing with SMB. It is often difficult to determine whether these are the product of normal traffic, misconfigured machines, or malicious intent. Ports 137, 138, and 139 should be blocked both in and out at your perimeter. There is no reason for this traffic to have to leave your local network. Since this topic has been dealt with extensively in previous evaluations, a detailed analysis was not undertaken.

Summary

As the many examples above indicate, your systems are subject to intense scrutiny from external networks. Your first line of defense should be firewalls. Hopefully, the traces we received were taken outside of any such firewalls. Besides the major examples above, there were any number of smaller scans, probes, and unidentified activities taking place. There are simply too many to analyze with the limited time and information provided. Perhaps with a better idea of the defenses your network currently has, we could ignore broad host and port scans and other attacks which you may not be vulnerable to, and instead focus on more realistic threats contained in the voluminous logs.

Return to top.

Analysis Methodology

The first step was to simply examine the data and get an idea of what I was looking at. It quickly

became clear the naming convention for the files was "SnortA*" for alert files, "SnortS*" for the scan reports, and "SOOS*" for packet details. The numbering of each series of files corresponds to temporal order, but there is no particular correspondence between the different types. To make the timing of the files a bit more clear, I renamed all files,

```
mmdd_original-name
```

The biggest problem with most IDS is not the lack of data, but rather false alarms. This data set is a clear example. It was quite obvious from from a cursory examination of the data that a substantial amount of the detects and alerts correspond to normal traffic. My approach to security is generally an explicit match to pass and then subject everything else to a default deny. I apply this to intrusion detection by taking the data, removing the traffic that I allow and expect, and then see what I have left.

However, this proves to be very difficult under these circumstances. Typically, when I am analyzing network traces I am familiar with the network or have some means to get more information about the network. This way, I can pull out false alarms and non-threatening traffic. For example, in the "Analyze This" data, if those machines being hit with all of SMTP traffic are the actual mail servers, there is not much cause for alarm. If they are not, we need to examine the SMTP traffic carefully. With no information, we cannot put it aside and focus on more "interesting" traces.

All of the port scans and host scans create a lot of noise in the logs. On my networks, I know what hosts are running services and how firewalls are protecting networks and hosts. When I see a scan bounce of my firewall, I can quickly determine the threat. I can record the source of the scan and any other pertinent information for posterity and then move on. But in this case I have no information about existing defenses, every kiddie's scan requires some level of analysis to watch for possible exploits. This consumed a large amount of effort.

We did have the previous practicals to use for reference. These practicals mostly focused on these noisy features in the logs. We can use these to verify that this has been an ongoing situation, but I still felt I needed to watch for changes from the patters they described.

All of those issues aside, I did my best to follow my security prefered model. I started with the most broad features of the logs, the various traffic from NCFC, isdn.net.il, *etc.* analyzed them and then removed them. In this way I moved to smaller and smaller features. Most of the external tools for Snort analysis I was able to find primarily focused on making data pretty, but did not do much for analysis. I mostly used UNIX command line tools like `grep`, `sort`, and `uniq`. Scripting languages like `awk` and `perl` were used heavily as well.

This script is an example of the quick-and-dirty tools I created to analyze the data. It was designed to reduce data on host scans. It parses a Snort scan log. It tracks all of the ports scanned on a destination host for a given IP source and destination pair. The original had no comments, but I have added some for clarification here,

```
#
# scanstat.awk
#
```

```
# Only parse lines that start with a date
/^(Aug|Sep)/ {
  # Break the source IP address from the port number (XXX.XXX.XXX.XXX:YYY
  split($4,ipport,":");
  # Store the source IP
  ips = ipport[1];
  # Break the destination IP from the port
  split($6,ipport,":");
  # Store both destination IP and port
  ipd = ipport[1];
  portd = ipport[2];
  # The index of the array is a string (gotta love AWK)
  indx = ips " -> " ipd;
  # Store the list of ports scanned in the array
  scan[indx] = scan[indx] " " portd;
}

# After the input is read, this is executed
END {
  # Iterate over the indicies of the array scan[]
  for ( indx in scan ) {
    # If the number of spaces in the string is greater than 4...
    if ( gsub(/ /,"&",scan[indx]) > 4 ) {
      # Print the source and destination IP (remember they are the
      # index) and the list of scanned ports
      print indx ": " scan[indx];
      # This is commented out, but sometimes wanted to print just the IPs
#      print indx;
    }
  }
}
```

The next example is a step-by-step demonstration of how I would isolate a certain feature of the traffic. In this case, I was removing the SNMP traffic for further analysis. First, I pull out the SNMP traffic,

```
$ grep ':161 *$' SnortA* > SNMP/port_161.A
```

Before I move on to analyze it, I remove SNMP from my "cleaned" alert file. The cleaned alert file is a concatenation of all of the alerts. Each time an alert is put aside for analysis, I remove it,

```
$ grep -v ':161 *$' snort_alert.clean > snort_alert.clean.0
$ mv snort_alert.clean.0 snort_alert.clean
```

When I want to move on to the next set of traces to analyze, I visually examine the clean file to see what interesting looking data remains. For this particular example, I had more analysis of the SNMP to do. I wanted to verify that no SNMP public queries were made from the outside,

```
$ grep -v 'MY\.NET\..* -> MY\.NET\.' SNMP/port_161.A
$ _
```

As I prefer, I used a explicit pass to remove traffic that I was not overly concerned about and

watched for anything else to fall out. In this case we were lucky and nothing did.

I still felt that using public SNMP internally was not a good idea, so I wanted to present that point in the above analysis. I checked earlier practicals to see if this was a new feature. It was not new. Since other practicals had gone in depth about why public SNMP strings are bad, I felt it best for me to skip a rehash of details and just give a summary of what hosts were still using it. I had noticed through visual examination that all of the SNMP traffic was destined for one host. For the report, I wanted one last bit of information from the logs, the list of machines that were making SNMP queries and the number of times they did it. At the command line I took full advantage of pipes to generate the HTML in one step,

```
 $ awk '{ sub(/:[0-9]*/,"",$7); print $7 }' SNMP/port_161.A | sort | uniq
    sort -nr | awk '{ print "<tr><td> " $2 " </td></td> " $1 " </td></tr>
<tr><td> MY.NET.98.172 </td></td> 226 </td></tr>
<tr><td> MY.NET.98.109 </td></td> 208 </td></tr>
<tr><td> MY.NET.97.217 </td></td> 194 </td></tr>
<tr><td> MY.NET.97.154 </td></td> 53 </td></tr>
<tr><td> MY.NET.98.114 </td></td> 52 </td></tr>
<tr><td> MY.NET.98.171 </td></td> 33 </td></tr>
<tr><td> MY.NET.98.191 </td></td> 31 </td></tr>
<tr><td> MY.NET.98.181 </td></td> 26 </td></tr>
<tr><td> MY.NET.97.244 </td></td> 23 </td></tr>
<tr><td> MY.NET.98.159 </td></td> 21 </td></tr>
<tr><td> MY.NET.98.201 </td></td> 19 </td></tr>
<tr><td> MY.NET.98.148 </td></td> 16 </td></tr>
<tr><td> MY.NET.97.246 </td></td> 6 </td></tr>
<tr><td> MY.NET.98.177 </td></td> 5 </td></tr>
<tr><td> MY.NET.97.206 </td></td> 5 </td></tr>
<tr><td> MY.NET.98.190 </td></td> 4 </td></tr>
$ _
```

I then continued this type analysis on the larger features in the logs. If the traffic looked threatening, I would add it to the analysis. If in my judgment the traffic was most likely a false alarm, I had to decide whether to present that information in the analysis or drop it silently. That is actually a difficult call for an assignment like this. I have the desire to tell the evaluator all of my findings, including why I ignored a substantial amount of traffic (e.g. patterns that appear to be valid DNS query replies), but in the pretext of the assignment, an analysis of a network's security for a bidding process, I would not include such extraneous information. I had to try to find a happy medium, and I hope I came close.

Return to top.

Appendix A

In order to create logs in the format shown in the network traces, Checkpoint Firewall-1 logs are "exported" to another machine for processing,

```
# /opt/CKPdw/bin/fw logexport -n -o /var/tmp/fwlog.exp
# scp /var/tmp/fwlog.exp process-host:Firewall/Processing
```

Various command line and script tools are then used to analyze the logs. The methods used are very similar to those outlined in the Analysis Methodology above.

A script was used to generate the traces as shown. The script contains code for several automatic log scans as well as a framework for doing more complicated analyses from the command line. Only a small portion of the script is actually important for just printing the logs. However, the entire script is presented for your perusal. Some specific host and network numbers have been obfuscated.

```
#
# fwlog.awk - 2000/10/24, cclark
#
# Template for processing FW-1 text logfiles. Based on the awk
# script for pulling apart exported logs.
#
# 2000/11/07- Seems the columns for "translated" logs and
# untranslated ones turn out different? ("Translated" means
# IP to hostnames.) Columns 17-20 swap with 21-25?

# $1  num
# $2  date
# $3  time
# $4  orig
# $5  type
# $6  action
# $7  alert
# $8  i/f_name
# $9  i/f_dir
# $10 proto
# $11 src
# $12 dst
# $13 service
# $14 s_port
# $15 len
# $16 rule
# $17 xlatesrc
# $18 xlatedst
# $19 xlatesport
# $20 xlatedport
# $21 agent
# $22 orig_from
# $23 orig_to
# $24 from
# $25 to
# $26 icmp-type
# $27 icmp-code
# $28 reason:
# $29 srckeyid
# $30 dstkeyid
# $31 user
# $32 scheme:
# $33 methods:
# $34 reason
# $35 error notification:
# $36 message
```

```
# $37 h_len
# $38 ip_vers
# $39 port:
# $40 ISAKMP Log:
# $41 sys_msgs

function spaceip(ip,   ipb) {
  if ( ip ~ /^[0-9\.]+$/ ) {
    split(ip,ipb,/\./);
    return sprintf("%3d.%3d.%3d.%3d",ipb[1],ipb[2],ipb[3],ipb[4]);
  } else {
    return ip;
  }
}

function stdprint(line) {
  split(line,w,";");
  dir = ( w[9] == "inbound" ) ? ">" : "<";
  printf "%9s %8s %6s %s%4s", w[2], w[3], w[6], dir, w[8];
  if ( ( w[10] == "tcp" ) || ( w[10] == "udp" ) ) {
    printf " %4s %s:%s -> %s:%s %d", w[10], w[11], w[14], w[12], w[13], w
  } else if ( w[10] == "icmp" ) {
    printf " %4s %s -> %s %d:%d %d", w[10], w[11], w[12], w[26], w[27], w
  } else {
    printf " %4s %s -> %s %d", w[10], w[11], w[12], w[15];
  }
  if ( w[17] ) {
    printf " (%s:%s -> %s:%s)", w[17], w[19], w[18], w[20];
  }
  printf "\n";
}

function fullprint() {
  print "\
       num: " $1 "\n\
      date: " $2 "\n\
      time: " $3 "\n\
      orig: " $4 "\n\
      type: " $5 "\n\
    action: " $6 "\n\
     alert: " $7 "\n\
  i/f_name: " $8 "\n\
   i/f_dir: " $9 "\n\
     proto: " $10 "\n\
       src: " $11 "\n\
       dst: " $12 "\n\
   service: " $13 "\n\
    s_port: " $14 "\n\
       len: " $15 "\n\
      rule: " $16 "\n\
  xlatesrc: " $17 "\n\
  xlatedst: " $18 "\n\
xlatesport: " $19 "\n\
xlatedport: " $20 "\n\
     agent: " $21 "\n\
 orig_from: " $22 "\n\
   orig_to: " $23 "\n\
```

```
      from: " $24 "\n\
        to: " $25 "\n\
 icmp-type: " $26 "\n\
 icmp-code: " $27 "\n\
  srckeyid: " $28 "\n\
  dstkeyid: " $29 "\n\
      user: " $30 "\n\
    reason: " $31 "\n\
    scheme: " $32 "\n\
   methods: " $33 "\n\
    reason: " $34 "\n\
error notification: " $35 "\n\
   message: " $36 "\n\
      port: " $37 "\n\
  sys_msgs: " $38 "\n";

}

BEGIN {
  if ( scan2 == 1 ) {
    FS="|";
    if ( slimit == 0 ) {
      slimit = 5;
    }
  } else {
    FS="\;";
  }
  i = 0;
}

( full == 1 ) && ( $5 == "log" ) && ( $8 != "daemon" ) {
  stdprint($0);
}

( mail == 1 ) && ( ( $13 == "smtp" ) || ( $25 == "mail" ) ) {
  fullprint();
}

( extdrop == 1 ) && ( $5 == "log" ) && ( $8 == "hme0" ) && ( $6 != "accep
  stdprint($0);
}

# UDP external traceroutes, dump slow DNS responses
( traceroute == 1 ) && ( $10 == "udp" ) && ( $8 == "hme0" ) && ( ( $13 +
  stdprint($0);
}

#
# Host scan checks. These are meant to be run in series
# since portions of the processing are best done outside
# of an AWK script (convert to perl?). Command line like,
#
# $ gzcat file.log.gz | awk -f fwlog.awk -v"scan1=1" | \
#     sort | awk -f fwlog.awk -v"scan2=1"

# Phase 1 of a check for scans
( scan1 == 1 ) && ( $8 == "hme0" ) && ( $9 == "inbound" ) && ( $6 != "acc
```

```
  print $11 "|" $12 "|" $13 "|" $0;
}

# Phase 2 of check for scans
( scan2 == 1 ) {
#  print "1: " w[12] "  " w[13];
  split($4,w,";");
  if ( ( ( w[12] == "XXX.XXX.248.142" ) && ( w[13] == "ident" ) ) \
       || ( ( w[12] ~ /(XXX.XXX.(248.147|152.1[05])|YYY.YYY.31.194)/ ) \
            && ( w[13] == "domain-udp" ) ) \
       || ( ( w[12] ~ /XXX.XXX.248.14[79]/ ) && ( w[13] ~ /^http/ ) ) \
       || ( ( w[12] ~ /XXX.XXX.152.1[123]/ ) && ( w[13] == 427 ) && ( w[1
       || ( ( w[12] ~ /XXX.XXX.15(2.1[123]|4.254)/ ) && ( w[13] == "ncp"
       || ( ( w[12] == "XXX.XXX.152.79" ) && ( w[13] ~ /(mail|pop-3|imap)
    next;
  }
#  print "2: " w[12] "  " w[13];
  if ( src == $1 ) {
    if ( i < slimit ) {
      i++;
      line[i] = $4;
    } else {
      if ( i > slimit ) {
        stdprint($4);
      } else {
        print "";
        for (j=1;j<=slimit;j++) {
          stdprint(line[j]);
        }
        stdprint($4);
        i++;
      }
    }
  } else {
    src = $1;
    i = 1;
    line[i] = $4;
  }
}

# Check for delayed mail
( chkmail == 1 ) && ( $13 == "smtp" ) {
#  stdprint($0);
  if ( $21 ~ /server/ ) {
    for(i=0;qmail[i];i++) { }
#    print i;
    qmail[i] = $0;
    if ( i > max ) {
      max = i;
    }
  } else { if ( $21 ~ /dequeuer/ ) {
    for(i=0;i<=max;i++) {
      split(qmail[i],pmail);
#      print pmail[11] " == " $11 "; " pmail[12] " == " $12 "; " pmail[14
      if ( ( pmail[11] == $11 ) && ( pmail[14] == $14 ) && \
           ( pmail[22] == $22 ) ) {
#       print "match!";
```

```
            if ( pmail[2] != $2 ) {
              print "--- Differing dates ---";
              stdprint(qmail[i]);
              fullprint();
            } else {
              split(pmail[3],otime,":");
              split($3,time,":");
              print "dt = " 3600*(time[1]-otime[1]) + \
                60*(time[2]-otime[2]) + time[3]-otime[3];
            }
            delete qmail[i];
          }
        }
        if ( qmail[i] ) {
          print "--- Not queued? ---";
          fullprint();
        }
      } }
    }

    END {
      if ( chkmail == 1 ) {
        for(i=0;i<=max;i++) {
          stdprint(qmail[i]);
        }
      }
    }

    #End
```

If you find the script useful or have any interesting modifications to improve it, please mail me and let me know.

As an example of how to use this, the logs for the first trace were generated by looking for entries in an archived (gzipped) log file with either a source or destination IP address of 128.100.71.72,

```
$ gunzip fw_20001105.log.gz | \
       awk -f fwlog.awk --source '( $11 == "128.100.71.72" ) || ( $12 ==
```

Return to top.