Mike Aylor
GCIA Practical v.3.2
Intrusion Detection in Depth
October 4, 2002

Part I - Describe the State of Intrusion Detection

Abstract

This paper centers around Ettercap and its use within an entirely switched network. Because of length limitations, this paper will assume some degree of competency from its audience, such as their familiarity with MAC addresses, switches and VLAN's, hubs, and ARP. While brief descriptions of some pertinent networking technologies will be given, all of these concepts are already very well documented in other sources as they represent many of the fundamentals in modern networking, and discussing them here in detail would be burdensome.

First, we will examine some of the differences between hubbed versus switched networks, and how this can affect network sniffing technologies. Second, we will discuss the concept of the arp cache, and then introduce Ettercap and how it works within a switched environment. Third, we will dissect a simulated Ettercap "attack". Fourth, we will discuss techniques for possibly detecting or preventing such an attack.

Introduction

All sniffer applications, like tcpdump, snoop, or Sniffer, work best when they are plugged into a network segment that can see all traffic. Hubs, by their nature, send any traffic received from one port to all other ports, thus allowing a sniffer plugged into that same hub to watch all conversations taking place between all hosts. Because so much network critical information is sent clear text (or with such weak encryption, it might as well be clear text), this means that the sniffer would also see this information even though they were not the intended recipient of the traffic. Worst case scenario (or best case, depending who you are), that means the sniffer would acquire passwords, bank account information, social security numbers, confidential email, anything that travels the wire would wind up in the log file of the sniffer.

Switches, on the other hand, do not blindly send all traffic to every port. They selectively forward traffic based upon a number of criterion, depending on the setup of the switch. A switch will associate ports with MAC addresses, and only forward those packets destined for those specific MAC addresses out that specific port. In some cases, ports can be grouped according to Virtual LAN's or VLAN's, and only ports associated with a common VLAN would be allowed to communicate with each other.

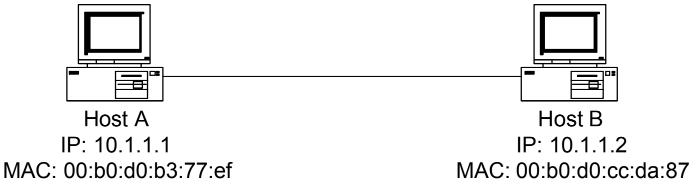
Broadcast traffic, such as ARP broadcasts, would be propagated to all ports of a common VLAN, but any replies to these broadcasts would be correctly forwarded only to that port that need to hear it. This behavior of switches has a number of effects. First, it can greatly improve the efficiency of a network: only those hosts who need to communicate can communicate. Hosts would be blissfully unaware of any communication that didn't somehow involve them.

Second, anyone who is using a sniffer on the switch would be unable to watch all communication. They would only be able to see broadcast traffic and traffic that their host was directly involved with.

However, some sniffer programs have found clever ways of getting around the switch's limitations and taking advantage of a machine's ARP cache.

ARP Cache

ARP, when used normally, is used to resolve unknown MAC addresses from known IP addresses. For example, suppose host A wanted to ping host B according to this diagram.



1. Host A would need to learn the MAC address of Host B. It would therefore generate an ARP broadcast querying for Host B's MAC Address, supplying the only bit of information it currently knows about Host B: it's IP Address.

2. Host B, along with any other host that happened to be on that network segment, would receive the broadcast query. Since the IP Address supplied in the ARP query includes Host B's IP Address, Host B responds with its MAC Address.

3. Host A would generate a icmp echo request packet with the MAC Address information it learned in the previous step.

4. Host B would respond with an icmp echo reply.

This is a tcpdump of all four steps combined (timestamps have been removed for readability):

```
00:b0:d0:b3:77:ef ff:ff:ff:ff:ff:ff 0806 42: arp who-has 10.1.1.2 tell 10.1.1.1
00:b0:d0:cc:da:87 00:b0:d0:b3:77:ef 0806 60: arp reply 10.1.1.2 is-at 00:b0:d0:cc:da:87
00:b0:d0:b3:77:ef 00:b0:d0:cc:da:87 0800 98: 10.1.1.1 > 10.1.1.2: icmp: echo request (DF)
00:b0:d0:cc:da:87 00:b0:d0:b3:77:ef 0800 98: 10.1.1.2 > 10.1.1.1: icmp: echo reply (DF)
```

Almost all PC's have what is known as a arp cache. This is a reserved space in the computer's memory that holds MAC Address to IP Address mappings, and it is constantly being updated with new information. In the previous example, Host A's arp cache was updated with Host B's information based on the second packet. Host B's arp cache was updated with Host A's information based on the FIRST packet (can be found at http://www.ietf.org/rfc/rfc0826.txt?number=826). It is important to bear in mind that the creators of the ARP protocol were primarily interested in providing a method by which hosts could change IP addresses and would have minimal impact on the network and connected hosts. For this reason, they specifically designed the ARP protocol to grab information wherever they could find it. To quote from RFC 826, "Notice that the <protocol type, sender protocol address, sender hardware address> triplet is merged into the table before the opcode is looked at. This is on the assumption that communication is bi-directional; if A has some reason to talk to B, then B will probably have some reason to talk to A."

The "opcode" the above quote refers to is the "REQUEST" or "REPLY" flag set into an ARP packet. This means that a host receiving any ARP packet regardless of whether it is a request or reply, regardless of whether it even requested ARP information from another host, it will update its arp cache. This is a wonderful means of maintaining an ARP table assuming that all devices on the network are behaving according to the LAN Administrator's wishes. This is also good for those who wish impersonate hosts on the network.

The technique of purposefully inserting incorrect associations of IP and MAC Addresses into a host's arp cache is known as arp poisoning. Generally, there are two reasons why this happens.

1. A hacker is attempting to create a condition which all hosts obtain incorrect ARP cache information, effectively preventing them from properly crafting packets. This results in a denial of service.

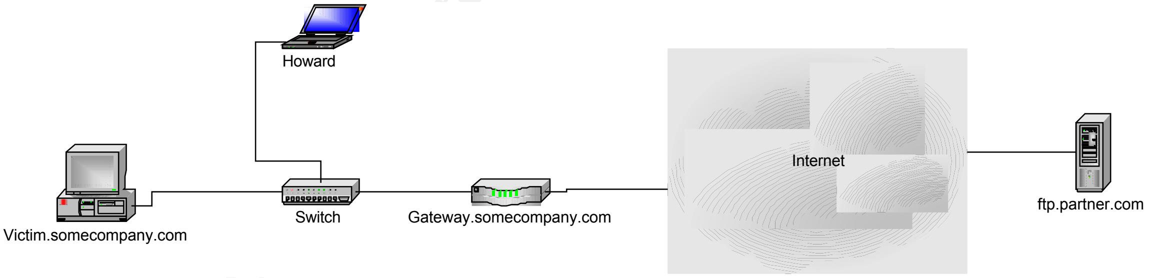
2. A hacker is attempting to insert themselves into the traffic stream, giving them the ability to watch (and possibly manipulate) all traffic as it passes through them.

<u>Ettercap</u>

Ettercap, which can be found at http://ettercap.sourceforge.net/, bills itself as a multipurpose sniffer/interceptor/logger for switched networks. It is not the only tool available to demonstrate arp poisoning style attacks, simply the one selected for this paper.

Ettercap relies upon the trusting nature of ARP caches to redirect traffic intended for other hosts to be instead sent to the Ettercap machine. By accomplishing this, it acts as the invisible middle man, causing all hosts involved in the conversation to think they are talking directly to one another, but are instead relaying all packets to the MAC Address of the host running Ettercap.

Let's see an example. Our unsuspecting victim wants to make a connection to a remote partner's FTP server to upload some files. Howard, a hired industrial spy, has snuck his laptop into the same switch as the victim and launched Ettercap. This diagram provides a rough approximation of the set up.



Howard sets his console to use arp poisoning, and to monitor all traffic between the victim and his default gateway. The victim launches an FTP client. The traffic below is a tcpdump taken from Howard's perspective (time stamps removed and packets numbered and spaced for readability).

```
1. MAC-HOWARD MAC-GATEWAY 0806 42: arp reply victim.somecompany.com is-at MAC-HOWARD
```

```
2. MAC-HOWARD MAC-VICTIM 0806 42: arp reply gateway.somecompany.com is-at MAC-HOWARD

3. MAC-VICTIM MAC-HOWARD 0800 62: victim.somecompany.com.3832 > ftp.partner.com.21: S
1752886802:1752886802(0) win 16384 <mss 1332,nop,nop,sackOK> (DF)

4. MAC-HOWARD MAC-GATEWAY 0800 62: victim.somecompany.com.3832 > ftp.partner.com.21: S
1752886802:1752886802(0) win 16384 <mss 1332,nop,nop,sackOK> (DF)

5. MAC-GATEWAY MAC-HOWARD 0800 60: ftp.partner.com.21 > victim.somecompany.com.3832: S
3997307881:3997307881(0) ack 1752886803 win 9324 <mss 1460> (DF)

6. MAC-HOWARD MAC-VICTIM 0800 60: ftp.partner.com.21 > victim.somecompany.com.3832: S
3997307881:3997307881(0) ack 1752886803 win 9324 <mss 1460> (DF)

7. MAC-VICTIM MAC-HOWARD 0800 60: victim.somecompany.com.3832 > ftp.partner.com.21: . ack 1 win
17316 (DF)

8. MAC-HOWARD MAC-GATEWAY 0800 60: victim.somecompany.com.3832 > ftp.partner.com.21: . ack 1 win
17316 (DF)
```

The first thing Ettercap does is send out an arp reply to both the gateway and the victim, informing each that the other is at MAC-HOWARD (packets 1 and 2). Note that neither the gateway or victim requested this information, but because of the trusting nature of ARP, each updates its ARP cache with the new information.

Now we see our victim begin by sending a SYN packet to ftp.partner.com (packet 3). Note that it does not send it to the default gateway's MAC address, but instead sends it to MAC-HOWARD. In packet 4, we see Ettercap generate the identical SYN packet, spoofing the victim's IP address but sending it to the MAC address of the gateway this time.

Packet 5 shows the FTP server's SYN ACK response, destined for MAC-HOWARD, and in packet 6, Ettercap turns right around and crafts the same packet and forwards it to the victim. Packets 7 and 8 show the completion of the TCP three way handshake, from the victim to Ettercap, from Ettercap to the gateway. The rest of the traffic showing the authentication and file upload (not shown here) takes place much the same way.

When Ettercap gets all the information it needs and is ready to relinquish control, it sends out an ARP reply with the "correct" MAC address information, thus taking itself out of the loop.

What's interesting to note is that Ettercap not only has complete visibility as to the conversation, but also that victim has no idea as to what has happened. Ettercap also has the ability to insert its own data into the traffic stream, and can intercept and decipher certain encrypted protocols (SSL, SSHv1) as well.

Detection and Prevention

Since Ettercap relies upon tricking layer 2 (ARP Cache's) into redirecting traffic, the key then is to focus on layer 2. Detecting such an attack would involve watching what hosts are advertising what MAC addresses. If an ARP reply gets sent saying that IP-A has MAC-1, then later another ARP reply is sent saying IP-A has MAC-2, and later still another ARP reply saying that IP-A is MAC-1, this is a good indicator that someone is misbehaving on the network. Either that or a zealous IT Help Desk analyst is doing a lot of troubleshooting with a network interface card on IP-A.

A pair of researches, Mahesh Tripunitara and Partha Dutta (http://www.acsac.org/1999/papers/fri-b-0830-dutta.pdf) describe the problem of ARP poisoning and have suggested adding a module that would intersect with ARP that would keep track of what MAC's a host has requested and whether it has received any responses. The theory here is that unsolicited ARP replies will be ignored and the ARP cache preserved. Only those ARP replies that were generated because of a prior request from the host would be permitted to modify the cache.

An idea I've considered is to statically set the ARP entry in every machine's ARP cache at least for that machine's default gateway. That would force it to only send traffic to a preset MAC address and would ignore any ARP's from other supposed gateways pretending to be the same IP. The drawbacks here is that it would be very difficult to ensure proper ARP setup on every host on a network, especially if the network has a heterogeneous mix of operating systems and requirements.

There are steps that can be taken to harden some switches, much like applying a layer 2 filter, permitting only certain MAC's to talk on certain ports. If a switch were to suddenly see packets from a different MAC, it could shut down the port and notify an administrator of the discrepancy.

Certain software applications are specially designed to detect changes in MAC address/IP address associations. For example, arpwatch, available from the Lawrence Berkeley National Laboratory (ftp://ftp.ee.lbl.gov/arpwatch.tar.Z), watches all ARP traffic on a given network, and will notify if a new station is detected, if an existing IP changed MAC addresses, and if a station changed from an old MAC to a new MAC back to the old MAC (flip-flop). I downloaded arpwatch and ran it to see if it would detect Ettercap poisoning the ARP cache in the above Ettercap example, this is one of many emails it generated.

```
From: Arpwatch@localhost.localdomain
Sent: Friday, July 12, 2002 9:28 AM
To: Michael Aylor
Subject: flip flop (gateway.somecompany.com)

            hostname: gateway.somecompany.com
          ip address: 10.1.0.1
    ethernet address: 0:0:c:08:ad:98
     ethernet vendor: Cisco
old ethernet address: 0:b0:d0:da:48:ad
 old ethernet vendor: Computer Products International
           timestamp: Friday, July 12, 2002 9:28:15 -0500
  previous timestamp: Friday, July 12, 2002 9:28:13 -0500
```

```
                    delta: 2 seconds
```

Conclusion

ARP caches are vulnerable. The designers of this system did not take into account the malicious packet crafter's who either want to deny a machine's ability to talk to it's neighbors, or insert themselves into a traffic stream for espionage purposes. While this risk is mitigated somewhat by the fact the hacker either has to have hacked a box on the local segment (i.e. same VLAN/collision domain) or has relocated themselves somewhere on the local segment, the risk is still present. Physical access to switch or hub ports can greatly reduce the possibility of an ARP poison attack

Note that if it is the administrator's goal to address the ARP poison denial of service rather than man-in-the-middle type attacks, physical layer security is not relevant. It would be fairly trivial for a hacker to write a worm that would cause an infected machine to constantly bombard its local LAN segment with erroneous ARP broadcasts, and they could accomplish this from the comfort of their easy chair.

Preventing ARP poisoning attacks can be accomplished by focusing on three critical areas

1. Physical Topology - where are the switches located and are they protected in secured locations? This includes the physical security of the hosts plugged into that switch.

2. Switch Configuration - does the switch allow for the administrator to set MAC address filtering or define rules that would cause a switch port to shut down if it detected MAC address irregularities?

3. Host's ARP Cache - can the ARP cache be statically locked with important hosts, such as gateways, database servers, domain controllers, etc? Can some third party module/application somehow make the ARP cache to be less trusting?

Today's firewalls and intrusion detection systems are inadequate for looking much into the frame layer of a packet and making useful decisions. There is certainly no quick fix to ARP poisoning attacks; none of the solutions presented would be easy and quick to implement or maintain. It is my feeling that designing/redefining the ARP protocol to only accept ARP solicited replies would be difficult to manage across a mix of platforms. It would involve writing separate drivers for each OS, and then pushing these changes to every host on a network (something that is not very easily accomplished in most networks). Statically setting the ARP cache of a machine has merit also, but again, it will require cooperation from every host on the network, and, if ever a MAC address changed (as in, if a default gateway router needed to be replaced and had a different MAC than the previous one), then the ARP caches would need to be updated or those hosts would risk losing all connectivity. Configuring switches to pay attention only to specific MAC addresses would be very cumbersome in medium to large size networks, since it would mean manually programming the MAC address of every host into a switch port, and updating the switch should ever any new hosts come online or existing one's changing network interface cards.

Perhaps addressing the ARP protocol at the RFC level would be beneficial, and motivating OS vendors to supply new code to support a new ARP RFC would be the best route to go. If necessity is the mother of invention, then perhaps it is only necessary to demonstrate the need for a new ARP RFC to those at the IETF.

Part II - Network Detects

Detect 1

Source of Trace

http://www.incidents.org/logs/Raw/2002.6.18

Detect was generated by:

Snort Intrusion Detection System v 1.8.7 using out of the box rules set.

Probability the source address was spoofed.

I would very much like to have this section at the bottom of my analysis, since my conclusion here is dependant on subsequent analysis not yet presented to the reader. For those of you reading this for the first time, I'd suggest skipping this part and coming back to it when you've read everything else.

If you've already read everything else in this write up, based on all the analysis I've done so far, I don't believe this to be a spoofed address. I believe the logs were obfuscated in such a way that I do not see all traffic, and I feel this was a false positive. An attacker would have little to gain by sending this traffic through the network, as it does not attempt to cause a denial of service, exploit some web vulnerability, or do any other known malicious activity. I believe this was an honest request that just happened to trigger a broadly written snort rule.

Description of Attack

Snort logged the following alert from the above tcpdump file.

```
[**] [1:1171:6] WEB-MISC whisker HEAD with large datagram [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/18-07:16:36.714488 202.214.44.44:1677 -> 46.5.180.133:80
TCP TTL:46 TOS:0x0 ID:60402 IpLen:20 DgmLen:570 DF
***AP*** Seq: 0x8A1E6C65  Ack: 0xDCAE637  Win: 0x4470  TcpLen: 20
[Xref =>
http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]
```

Here, the alert makes reference to "Whisker", which is an intrusion detection evasion tool created by the programmer Rain Forest Puppy (details of which can be found at http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html) . The premise of Whisker is that it uses atypical HTTP requests to fool IDS's. In this case, "whisker HEAD with large datagram" indicates that the packet is suspicious because it is seemingly a HEAD request but with a significantly large payload.

Typically, most IDS's will have rules that look for packets with malicious GET requests. Whisker defeats these basic rules by replacing GET requests with HEAD requests. So what are the differences between the two? For this, we must look at a number of sources.

From RFC 2616, Section 9.4 (found at http://www.ietf.org/rfc/rfc2616.txt?number=2616), which governs HTTP 1.1 HEAD methods, the HEAD method is the same as any GET request with only one exception. When the server receives a HEAD request, it is only supposed to respond with meta-information. Any actual message-body must be omitted from the server's response. To quote the RFC, "this method is often used for testing hypertext links for validity, accessibility, and recent modification." So, since there isn't supposed to be any actual data transferred during such a request, then we've got nothing to worry about, right?

Well, not quite. Even if the server responds with meta information, its still telling the attacker that the request was valid (per the RFC, the attacker would still receive an "HTTP 200" or something similar along with other tidbits of information if everything was okay, in theory.) Furthermore, in some cases, an attacker is not interested in pulling information from the server, but rather inserting information within, say, the default.htm file (such as in a website defacement). While the RFC covers that a server should respond with limited meta information, it makes no mention about how the server should treat the request, or, if the request indicated the web server should run a certain program, that that program wouldn't actually run.

Therefore, it is necessary to look into the packet payload to determine more about the type of attack this is. Tcpdump supplied the following information (I have bolded the packet information I was most interested in).

```
0x0000   4500 0239 efb0 4000 2e06 8787 cad6 2c2c        E..9..@.......,,
0x0010   2e05 b485 069d 0050 9803 3c83 45eb 5987        .......P..<.E.Y.
0x0020   5018 4470 f6c1 0000 4845 4144 202f 6d61        P.Dp....HEAD./ma
0x0030   696e 2f74 6f6f 6c73 2f69 6f2d 7363 682f        in/tools/io-sch/
0x0040   3637 782e 7064 6620 4854 5450 2f31 2e31        67x.pdf.HTTP/1.1
0x0050   0d0a 5669 613a 2031 2e31 202d 2028 4465        ..Via:.1.1.-.(De
0x0060   6c65 4761 7465 2f37 2e35 2e33 290d 0a48        leGate/7.5.3)..H
0x0070   6f73 743a 2077 7777 2e58 5858 582e 636f        ost:.www.XXXX.co
0x0080   6d0d 0a55 7365 722d 4167 656e 743a 204d        m..User-Agent:.M
0x0090   6f7a 696c 6c61 2f35 2e30 2028 5769 6e64        ozilla/5.0.(Wind
0x00a0   6f77 733b 2055 3b20 5769 6e64 6f77 7320        ows;.U;.Windows.
0x00b0   4e54 2035 2e31 3b20 6a61 2d4a 503b 2072        NT.5.1;.ja-JP;.r
0x00c0   763a 312e 3072 6332 2920 4765 636b 6f2f        v:1.0rc2).Gecko/
0x00d0   3230 3032 3035 3132 204e 6574 7363 6170        20020512.Netscap
0x00e0   652f 372e 3062 310d 0a41 6363 6570 743a        e/7.0b1..Accept:
0x00f0   2074 6578 742f 786d 6c2c 6170 706c 6963        .text/xml,applic
0x0100   6174 696f 6e2f 786d 6c2c 6170 706c 6963        ation/xml,applic
0x0110   6174 696f 6e2f 7868 746d 6c2b 786d 6c2c        ation/xhtml+xml,
0x0120   7465 7874 2f68 746d 6c3b 713d 302e 392c        text/html;q=0.9,
0x0130   7465 7874 2f70 6c61 696e 3b71 3d30 2e38        text/plain;q=0.8
0x0140   2c76 6964 656f 2f78 2d6d 6e67 2c69 6d61        ,video/x-mng,ima
0x0150   6765 2f70 6e67 2c69 6d61 6765 2f6a 7065        ge/png,image/jpe
0x0160   672c 696d 6167 652f 6769 663b 713d 302e        g,image/gif;q=0.
0x0170   322c 7465 7874 2f63 7373 2c2a 2f2a 3b71        2,text/css,*/*;q
0x0180   3d30 2e31 0d0a 4163 6365 7074 2d4c 616e        =0.1..Accept-Lan
0x0190   6775 6167 653a 206a 615f 4a50 2c20 6a61        guage:.ja_JP,.ja
0x01a0   3b71 3d30 2e35 300d 0a41 6363 6570 742d        ;q=0.50..Accept-
0x01b0   456e 636f 6469 6e67 3a20 677a 6970 2c20        Encoding:.gzip,.
0x01c0   6465 666c 6174 652c 2063 6f6d 7072 6573        deflate,.compres
0x01d0   733b 713d 302e 390d 0a41 6363 6570 742d        s;q=0.9..Accept-
0x01e0   4368 6172 7365 743a 2053 6869 6674 5f4a        Charset:.Shift_J
0x01f0   4953 2c20 7574 662d 383b 713d 302e 3636        IS,.utf-8;q=0.66
0x0200   2c20 2a3b 713d 302e 3636 0d0a 5072 6167        ,.*;q=0.66..Prag
0x0210   6d61 3a20 6e6f 2d63 6163 6865 0d0a 4361        ma:.no-cache..Ca
0x0220   6368 652d 436f 6e74 726f 6c3a 206e 6f2d        che-Control:.no-
0x0230   6361 6368 650d 0a0d 0a                         cache....
```

This was a request for "HEAD /main/tools/io-sch/67x.pdf" according to the packet payload. In my experience, most such exploits call either .exe's, /cgi-bin/, or some executable somewhere buried within the web server's directory structure. In some cases they'll attempt to insert their own binary code into the web server's memory stack, as in the case of a buffer overflow attack (such as Nimda).

I personally have never heard of any exploit sounding anything remotely like this, so I did some web searches for various parts of this path. It turns out that /main/tools/io-sch/67x.pdf is a real live url referencing an actual PDF file. According to the packet payload, the host is www.XXXX.com. I presume that the people who made this packet trace publicly available attempted to obfuscate the actual host of this trace, and even though I discovered the true host of this .pdf document, I will respect their privacy and leave that out of my analysis. Suffice it to say, doing a simple nslookup on the destination address does not yield the host I discovered that houses this .pdf document, so presumably the destination address in this trace was also obfuscated (makes sense really, why go to the trouble of obfuscating the host information and the not IP address information?) Despite the fact that snort triggered an alert, I believe this alert is a false positive.

Attack mechanism

As mentioned above, I believe this attack to be a false positive. However, if this had been an actual attack, it would work with the attacker sending a specially formatted HEAD request in an effort to gain useful information about the victim web server, while at the same time evading common IDS signatures. The significant part of this is not that the attacker would be attempting to compromise the machine, but that the attacker is also attempting to cover his trail by not tipping off any IDS watching him/her. The rule catching this attempt flagged this as a Whisker style attack, which is a thoroughly documented anti-IDS tool within the alert url.

Correlations

Since this was not actually an attack, I have no other log data to review other than the original tcpdump file. But sometimes even the lack of information is a correlation.

The source and destination of this alert had only two packets in the entire trace.

```
07:16:36.714488 202.214.44.44.1677 > 46.5.180.133.80: P
2317249637:2317250167(530) ack       231401015 win 17520 (DF)
07:31:24.894488 202.214.44.44.1693 > 46.5.180.133.80: P
2550348931:2550349460(529) ack       1173051783 win 17520 (DF)
```

There is no evidence of any TCP three way handshake, no sign of any acknowledgement from the web server that it received this HEAD requests, for that matter, no sign that the target of the HEAD request is even listening on port 80. I can only surmise one of two possibilities.

1. This trace is incomplete and purposefully truncated to make it appropriate for public consumption.

2. An attacker sent this packets without actually expecting to get the .pdf document in question. Possibly to mask some other activity, possibly to cause snort to trigger the HEAD alert and mislead the IDS analyst.

3. The sniffer at this network segment was programmed to filter out 3 way handshake traffic.

Combine this with the fact that the path of the HEAD request is an actual .pdf document, and that the HEAD request makes no mention of any executable or binary, I continue to claim this was a false positive, leaning more towards explanation 1 rather than 2 or 3.

Evidence of Active Targeting

This was not a broad sweep as is typical in a port scan. This is probably not a wrong number case either. This was very likely targeted to us on purpose.

Severity

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

Criticality: Assuming the target really is a web server that really does have this .pdf document on it, then presumably this server is important to the company. If this were the company's sole Internet presence, and if the company's business relied heavily on having a reliable reputation, then this rates as a 5.

Lethality: The attack was actually a false positive. 0

System Countermeasures: I have little information pertaining to the security strength of the destination. If it isn't hardened or patched against the very latest threats, actual attacks will succeed against it. However, based strictly on the traces I have, this system is so hardened, it might as well be offline (nowhere do I see it talk to anything). 5

Network Countermeasures: Again, very little information here. Again, based on the logic used in the System Countermeasures section, the host might as well be non-existent. 5

Severity = (5 + 0) - (5 + 5) = -5

Defensive Recommendations:

None. No evidence that anything improper is happening.

Test Question:

You receive the following packet trace in your logs and your IDS has flagged these packets as suspicious.

```
07:16:36.714488 202.214.44.44.1677 > 46.5.180.133.80: P
2317249637:2317250167(530) ack       231401015 win 17520 (DF)
07:31:24.894488 202.214.44.44.1693 > 46.5.180.133.80: P
2550348931:2550349460(529) ack       1173051783 win 17520 (DF)
```

These are the only packets with these source and destination addresses. You notice that there is no reference to any TCP three way handshake as you might expect in legitimate TCP traffic. A colleague mentions to you that you might want to make sure there isn't a routing loop somewhere in your network, and that you're probably only seeing half the conversation. What reason can you give him that would negate this explanation?

A. If there were a routing loop, you should see ICMP - "Multiple Path to Destination" messages originating from your router.
B. If there were a routing loop, you should still see parts of the TCP three way hand shake, specifically, a SYN packet from 202.214.44.44 and an ACK packet from the same. Presumably, the second SYN ACK packet would have been looped to a different path. (THIS IS THE ANSWER)
C. None of the above.
D. There is no such thing as a routing loop.
E. Routing Loops is actually a breakfast cereal.

Q&A from Mailing List

This detect was submitted to the instrusion@incidents.org mailing list for comment. Listed below are top 3 questions posed by responders...

1. From Chris Benton on 8/5/2002

```
> 0x01a0   3b71 3d30 2e35 300d 0a41 6363 6570 742d        ;q=0.50..Accept-
> 0x01b0   456e 636f 6469 6e67 3a20 677a 6970 2c20        Encoding:.gzip,.
> 0x01c0   6465 666c 6174 652c 2063 6f6d 7072 6573        deflate,.compres
```

Interesting. A Windows machine that supports gzip but not zip. Hummm....

Response: From another responder, Barry Doran...

```
> Interesting. A Windows machine that supports gzip but not
```

> zip. Hummm....

Actually no, IE standard header

HTTP_ACCEPT_ENCODING = gzip, deflate

http://www.bann.co.uk/asp/simple/scripts/debug.asp will dump them all
the passed HTTP headers out if you wish to confirm.

This is actually a fingerprint of a Windows 2000 box, that it uses gzip encoding rather than zip. Not necessarily and indicator that something is out of spec.

2. This lead to a follow-up question that I posed to Chris regarding the importance of knowing the OS of your attacker. What follows is my questions noted with ">"'s and his response.

> How useful is it to determine the OS of the host? I've read a few papers on
> passive OS fingerprinting, but I've never been sold on the idea. My
> argument is that the attack itself is the only important aspect, what is
> someone trying to do to your boxes...why is the platform they run it off of
> relevant?

Guess it depends on how much you want to get into it. If all you are interested in is "gee I got attacked" and you plan on dropping the matter there, then I would agree. If you want to take it to the next level however, knowing the source OS can be extremely helpful. This gives you a wealth of info including attacker skill set, available tools, weed out false positives, etc.

I'll give you an example, let's say you receive a few suspicious packets that have one or both of the high order bits set in TCP byte 13. Many IDS boxes still flag this as an attack. Should you be concerned? You really have no way of knowing unless you can fingerprint the source OS. If it's Linux running a 2.4 kernel, it's probably legitimate traffic as ECN support is built right in. If the box fingerprints as Win2K, Win98, etc. then it's probably someone probing your network as Windows does not yet support ECN.

So if you plan on ignoring your IDS when it goes off, then the source OS does not matter. If you plan on following up on the alert, you really want to fingerprint the source OS so you don't end up blocking someone needlessly or come off sounding like you don't know what you are doing if you report it.

> Plus, these attack scripts are getting so sophisticated, they can probably
> mask their OS characteristics to pretend to be another OS. Your thoughts?

Which can allow you to fingerprint the tool they are using as well. Check out the p0f database. I know there is at least entries for nmap and Queso.

HTH,

C

My Response to Chris:
I like this example of the high order bit for the ECN flag, because its something I ran across when attempting to compile linux kernel 2.4.0 when it was first released. I accidentally compiled it with ECN support and I was unable to communicate with devices outside the firewall because the firewall thought the ECN was out of spec, and thus dropped it. But enough history.

Its oversimplification to call my strategy "gee I got attacked" and then drop the issue. I equate this to triage during a war. Suppose a soldier were hit by a rifle bullet. I would want to know how badly the soldier was wounded, where he was wounded, perhaps even the type of bullet was used. I might want to know where the soldier was when he was hit so I could gauge where the enemy was and take precautions for preventing any more soldiers being wounded. In fact, if I were so inclined, I could serialize every bullet I'd extracted from a soldier, send it through a gas-spectrometer to get its exact shell material composition, look for patterns in the blood type of all soldiers wounded to determine a possible pattern, the number of details I could investigate is virtually limitless. The scarcest resource of any analyst is time: time to analyze, recommend, implement, and ultimately thwart an attack. Time should be spent wisely and not in the pursuit of what amounts to trivia.

If I spent my time determining the most miniscule details about every bullet that came my way, I might lose some patients. Determining the OS of an attacker would be useful if I were planning on prosecuting them in court, but I don't think there's enough psycho-social evidence to suggest that an attacker who originates an attack from a linux box is more "sophisticated" than one from a Windows box. In fact, many attacks launched from either are sometimes drones anyway, so the OS would only mislead the analyst from the one single simple truth that matters to his/her employers: We're under attack and what are we going to protect ourselves? Do we implement security policies and practices based on the OS of our attacker? All decent IDS signatures are based on the signature of the actual attack. I still feel that the OS qualifies as "trivia" and little else to the business at hand. Only if the OS were relevant to determining the type of attack would I consider it to be useful knowledge (i.e. a new worm that propagates from source hosts with the ECN flag set).

3. From Chris, posted on 8/5/2002

```
> The source and destination of this alert had only two packets in the entire
> trace.
>
> 07:16:36.714488 202.214.44.44.1677 > 46.5.180.133.80: P
> 2317249637:2317250167(530) ack       231401015 win 17520 (DF)
> 07:31:24.894488 202.214.44.44.1693 > 46.5.180.133.80: P
> 2550348931:2550349460(529) ack       1173051783 win 17520 (DF)
```

This is pretty weird. A couple of possibilities:
1) Three way was prior to trace capture, above packets are being dropped

by a firewall

2) Three way prior to trace capture, type 3 returned by some other IP address

3) Packets were crafted in a test environment

I'm inclined to agree with #3 at this point. There's a 15 minute spread between pushes from this host, but on different sockets (notice the source ports are different for each packet). I don't agree with item 1 or 2 because the trace log starts at about 19:00 the previous day, and you don't commonly see TCP sessions last that long (most TCP stacks would have timed out the session by that point anyway).

Detect 2

Source of Trace

http://www.incidents.org/logs/Raw/2002.6.13

Detect was generated by:

Snort Intrusion Detection System v 1.8.7 using out of the box rules set.

Probability the source address was spoofed.

Unlikely. This was an application layer recon scan, implying that some kind of response is necessary to make this kind of activity worth the attacker's while. In addition, in order to successfully query for _vti_inf or _vti_rpc, it is first necessary to establish a TCP session with the victim. While it is technically possible for an attacker to predict TCP sequence numbers and spoof a source address, it would be more complicated. In this attack, I only see one source address. If an attacker were spoofing an address, they might have greater difficulty in seeing the response traffic (it could be possible if the attacker were spoofing an address on the same subnet and collision domain as a host he already has control over, thus, while the responses would be directed to him specifically, he should be able to sniff the wire to see the responses).

Again, spoofing is possible, but not likely in this analyst's opinion.

Description of Attack

Snort logged the following alerts when run against this tcpdump file.

```
 [**] [1:990:5] WEB-IIS _vti_inf access [**]
[Classification:  sid] [Priority: 2]
07/12-19:34:49.644488 203.241.151.50:60592 -> 46.5.180.133:80
TCP TTL:48 TOS:0x0 ID:16760 IpLen:20 DgmLen:314
***AP*** Seq: 0xE472925D  Ack: 0x7EAF47A4  Win: 0xFFFF  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4461199 7368629

[**] [1:937:6] WEB-FRONTPAGE _vti_rpc access [**]
[Classification:  sid] [Priority: 2]
07/12-19:34:50.944488 203.241.151.50:61193 -> 46.5.180.133:80
TCP TTL:48 TOS:0x0 ID:24172 IpLen:20 DgmLen:441
***AP*** Seq: 0xCA1B00C8  Ack: 0x7EC45A56  Win: 0xFFFF  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4461202 7368758
[Xref => http://www.securityfocus.com/bid/2144]
```

There are two alerts here, one for "WEB-IIS _vti_inf access" and "WEB-FRONTPAGE _vti_rpc access". Both come from the same source to the same destination all seemingly over http.

_Vti_inf.html is loaded as part of the IIS implementation of Microsoft FrontPage. FrontPage was designed as a means for web designers to easily generate and administer web page content, and includes a number of "features" such as the ability to remotely push changes to any given web page. _Vti_inf.html handles the server's base information, such as version number and scripting paths. An example _vti_inf.html for us to review…

```
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<title> FrontPage Configuration Information </title>
</head>
<!-- _vti_inf.html version 0.100>
<!--
This file contains important information used by the FrontPage client
(the FrontPage Explorer and FrontPage Editor) to communicate with the
FrontPage server extensions installed on this web server. The values
below are automatically set by FrontPage at installation. Normally,
you do not need to modify these values, but in case you do, the parameters
are as follows:
'FPShtmlScriptUrl', 'FPAuthorScriptUrl', and 'FPAdminScriptUrl'
specify the relative urls for the scripts that FrontPage uses for remote
authoring. These values should not be changed.
'FPVersion' identifies the version of the FrontPage Server
Extensions installed, and should not be changed.
--><!-- FrontPage Configuration Information
FPVersion="4.0.2.2717"
FPShtmlScriptUrl="_vti_bin/shtml.exe/_vti_rpc"
FPAuthorScriptUrl="_vti_bin/_vti_aut/author.exe"
FPAdminScriptUrl="_vti_bin/_vti_adm/admin.exe"
-->
<!--webbot bot="PurpleText" preview="This page is placed into the
root directory of your FrontPage web when FrontPage is installed. It
contains information us ed by the FrontPage client to communicate with
the FrontPage server extensions installed on this web server. You should
not delete this file." -->
```

Even though this looks like a recon scan, it is my opinion that if an attacker were to gain this knowledge, they would know your web server was possibly susceptible to FrontPage style attacks, and could result in a more targeted attack. It is like those who send out DNS “version.bind” requests in the hopes of finding vulnerable and exploitable BIND services. Such reconnaissance should not be underestimated. To quote Sun Tzu from The Art of War, “Know your enemy and know yourself; in a hundred battles, you will never be defeated.” _Vti_inf requests are designed to help the enemy know you.

The second part of this attack, referencing “_vti_rpc.html”, looks very similar to a documented problem with Microsoft IIS 4.0 and 5.0. According to eEye Digital Security (http://lists.insecure.org/win2ksecadvice/2001/Jan/0010.html), if someone were to sending the following text to a server, it would cause it to crash.

```
POST /_vti_bin/shtml.dll/_vti_rpc HTTP/1.1
Date: Mon, 23 Oct 2000 16:04:17 GMT
MIME-Version: 1.0
User-Agent: MSFrontPage/4.0
Host: 192.168.1.168
Accept: auth/sicily
Content-Length: 2000
Content-Type: application/x-www-form-urlencoded
X-Vermeer-Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
<enter>
<enter>
[server returns data]
<enter>
<enter>
method=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA<lots of “A”’s, removed for readability>AAAAAAAA
```

Let’s compare this text with what was actually sent in from our attacker…

```
19:44:57.184488 203.241.151.50.16446 > 46.5.180.133.80: P 3511070194:3511070583(389) ack
2769305419 win 65535 <nop,nop,timestamp 4462415 7429379>
0x0000   4500 01b9 abc8 0000 3006 9dce cbf1 9732        E.......0......2
0x0010   2e05 b485 403e 0050 d146 adf2 a510 3f4b        ....@>.P.F....?K
0x0020   8018 ffff 91e1 0000 0101 080a 0044 174f        .............D.O
0x0030   0071 5d03 504f 5354 202f 5f76 7469 5f62        .q].POST./_vti_b
0x0040   696e 2f73 6874 6d6c 2e65 7865 2f5f 7674        in/shtml.exe/_vt
0x0050   695f 7270 6320 4854 5450 2f31 2e30 0d0a        i_rpc.HTTP/1.0..
0x0060   4461 7465 3a20 5361 742c 2031 3320 4a75        Date:.Sat,.13.Ju
0x0070   6c20 3230 3032 2030 313a 3535 3a34 3920        l.2002.01:55:49.
0x0080   474d 540d 0a4d 494d 452d 5665 7273 696f        GMT..MIME-Versio
0x0090   6e3a 2031 2e30 0d0a 5573 6572 2d41 6765        n:.1.0..User-Age
0x00a0   6e74 3a20 4d53 4672 6f6e 7450 6167 652f        nt:.MSFrontPage/
0x00b0   342e 300d 0a48 6f73 743a 2077 7777 2e58        4.0..Host:.www.X
0x00c0   5858 582e 636f 6d0d 0a41 6363 6570 743a        XXX.com..Accept:
0x00d0   2061 7574 682f 7369 6369 6c79 0d0a 436f        .auth/sicily..Co
0x00e0   6e74 656e 742d 4c65 6e67 7468 3a20 3431        ntent-Length:.41
0x00f0   0d0a 436f 6e74 656e 742d 5479 7065 3a20        ..Content-Type:.
0x0100   6170 706c 6963 6174 696f 6e2f 782d 7777        application/x-ww
0x0110   772d 666f 726d 2d75 726c 656e 636f 6465        w-form-urlencode
0x0120   640d 0a58 2d56 6572 6d65 6572 2d43 6f6e        d..X-Vermeer-Con
0x0130   7465 6e74 2d54 7970 653a 2061 7070 6c69        tent-Type:.appli
0x0140   6361 7469 6f6e 2f78 2d77 7777 2d66 6f72        cation/x-www-for
0x0150   6d2d 7572 6c65 6e63 6f64 6564 0d0a 5072        m-urlencoded..Pr
```

```
0x0160   6167 6d61 3a20 6e6f 2d63 6163 6865 0d0a        agma:.no-cache..
0x0170   4361 6368 652d 436f 6e74 726f 6c3a 206d        Cache-Control:.m
0x0180   6178 2d73 7461 6c65 3d30 0d0a 0d0a 6d65        ax-stale=0....me
0x0190   7468 6f64 3d73 6572 7665 722b 7665 7273        thod=server+vers
0x01a0   696f 6e25 3361 3425 3265 3025 3265 3225        ion%3a4%2e0%2e2%
0x01b0   3265 3236 3131 0a0d 0a                         2e2611...
```

There are no “A”s in the method, instead there is “server version”. This looks very much like another recon attempt to get information about the web server, not an attempted DOS.

Attack mechanism

The following is a brief summary of all packets to and from the attacker…

```
19:34:49.644488 203.241.151.50.60592 > 46.5.180.133.80: P 3832713821:3832714083(262) ack
2125416356 win 65535 <nop,nop,timestamp 4461199 7368629>
19:34:50.944488 203.241.151.50.61193 > 46.5.180.133.80: P 3390767304:3390767693(389) ack
2126797398 win 65535 <nop,nop,timestamp 4461202 7368758>
19:44:18.214488 203.241.151.50.60802 > 46.5.180.133.80: P 475005787:475006049(262) ack 2726606846
win 65535 <nop,nop,timestamp 4462337 7425481>
19:44:19.334488 203.241.151.50.61384 > 46.5.180.133.80: P 1493364766:1493365155(389) ack
2720916284 win 65535 <nop,nop,timestamp 4462339 7425594>
19:44:56.054488 203.241.151.50.15763 > 46.5.180.133.80: P 3203231433:3203231695(262) ack
2760327168 win 65535 <nop,nop,timestamp 4462412 7429266>
19:44:57.184488 203.241.151.50.16446 > 46.5.180.133.80: P 3511070194:3511070583(389) ack
2769305419 win 65535 <nop,nop,timestamp 4462415 7429379>
```

Given that the packets come so very close together with respect to time difference, it is likely this was an automated scanner. There are many cgi type scanners available, too many to list here.

Correlations

Since this traffic was made available from an anonymous third party contributor, I have no correlating logs. However, it’s worth noting that there were no response traffic packets to these attacks. Either the IDS sensor was not set up to see the response or there was no response.

Evidence of Active Targeting

This is not legitimate traffic. This is a recon scan. There is no indication of a TCP three way handshake, but I surmise there had to have been one, we’re just not showing it in the logs for some reason. In order for the recon to be successful, it would likely have had to establish a TCP connection before sending the GETS/POSTS.

They targeted only one host in this detect, but there is no indication they singled this host out (no sweeps of the network over port 80). Very likely they were just looking for targets of opportunity.

Severity

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

Criticality: a web server, presumably. No data about the web server was provided, no knowledge as to what kind of information is on it. Assuming worst case scenario (better to err on the side of caution). 5

Lethality: 2 Not really damaging, just a recon scan at this point, but not unimportant.

System Countermeasures: According to logs, no response was forthcoming. I cannot hazard a guess as to why. The server needs to be patched and have the FrontPage server extensions removed if not needed. 5

Network Countermeasures: Again, no knowledge pertaining to network configuration. Presumably, port 80 is necessary to this server, yet no sign of any return traffic. Presumably it was blocked. 5

Severity = -3

Defensive Recommendations:

Without more information about the server and network, I can't make recommendations other than the usual. Patch the server, remove FPSE if not needed. Turn off web services if not needed. Turn off the server if not needed.

Test Question:

Q: What does VTI in _vti_inf stand for?

A: Vermeer Technology Incorporated – the folks who originally wrote FrontPage and were later bought out by Microsoft.

<u>Detect 3</u>

Source of Trace

http://www.incidents.org/logs/Raw/2002.6.13

Detect was generated by:

Snort Intrusion Detection System v 1.8.7 using out of the box rules set.

Probability the source address was spoofed

Actually, the odds are fairly good that at least some of the addresses were spoofed. This type of scan was done very slowly from multiple sources, so the odds that the real attacker was trying to hide themselves amongst the other decoys is, in my opinion, fairly good.

Description of the Attack

The following is one of many SCAN SYN FIN alerts generated by snort...

```
[**] [1:624:1] SCAN SYN FIN [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/06-20:26:26.694488 166.104.219.69:21 -> 46.5.248.204:21
TCP TTL:19 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x4D10FE66  Ack: 0x4BFEFC88  Win: 0x404  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS198]
```

This was a recon scan for port 21 using what is known as a SYN/FIN scan. The idea is that the attacker is making improper combinations of TCP flags. Under no circumstances would a TCP session include a packet with both the SYN and FIN flags set; it is usually one or the other. However, by combining these flags together, the attacker could possibly get around certain some IDS signatures which are only looking for TCP scans with the SYN flag set.

So, what does a SYN/FIN scan accomplish? Well, it depends on the target OS, however, just to see what it looked like, I downloaded hping2 and crafted some SYN/FIN packets of my own and targeted them against one of my Windows 2000 web servers. Notice how the web server responded to the initial SYN/FIN packet...

```
command: hping2 Victim -SF -p 80
```

```
12:38:05.999226 Attacker.2420 > Victim.80: SF 701414086:701414086(0) win 512
12:38:05.999465 Victim.80 > Attacker.2420: S 2290677134:2290677134(0) ack 701414087 win 64240
<mss 1460> (DF)
12:38:05.999503 Attacker.2420 > Victim.80: R 701414087:701414087(0) win 0 (DF)
```

```
12:38:06.996339 Attacker.2421 > Victim.80: SF 1602231200:1602231200(0) win 512
12:38:06.996564 Victim.80 > Attacker.2421: S 2290957235:2290957235(0) ack 1602231201 win 64240
<mss 1460> (DF)
12:38:06.996579 Attacker.2421 > Victim.80: R 1602231201:1602231201(0) win 0 (DF)
12:38:07.996327 Attacker.2422 > Victim.80: SF 854233405:854233405(0) win 512
12:38:07.996545 Victim.80 > Attacker.2422: S 2291242244:2291242244(0) ack 854233406 win 64240
<mss 1460> (DF)
12:38:07.996562 Attacker.2422 > Victim.80: R 854233406:854233406(0) win 0 (DF)
```

Note that the victim responded with a SYN/ACK, just like it would if it had received a regular SYN packet. Thus, an attacker would be able to completely map open TCP ports of a given host.

I created a tcpdump filter to look for all SYN/FIN combinations in the trace log...

```
ip and ((tcp[13] & 2 != 0) and (tcp[13] & 1 !=0))
```

...and ran it against the 2002.6.7 file. These are the results...

```
19:47:38.094488 166.104.219.69.21 > 46.5.108.98.21: SF 2011671649:2011671649(0) win 1028
19:48:26.694488 62.153.209.202.21 > 46.5.44.105.21: SF 537077617:537077617(0) win 1028
20:15:42.694488 166.104.219.69.21 > 46.5.225.12.21: SF 1728279025:1728279025(0) win 1028
20:26:26.694488 166.104.219.69.21 > 46.5.248.204.21: SF 1292959334:1292959334(0) win 1028
20:41:06.114488 62.153.209.202.21 > 46.5.14.196.21: SF 491405744:491405744(0) win 1028
20:43:12.874488 62.153.209.202.21 > 46.5.249.83.21: SF 821607003:821607003(0) win 1028
20:43:15.134488 166.104.219.69.21 > 46.5.68.2.21: SF 1834095836:1834095836(0) win 1028
20:44:46.494488 62.153.209.202.21 > 46.5.242.253.21: SF 1226465352:1226465352(0) win 1028
20:50:02.424488 62.153.209.202.21 > 46.5.12.64.21: SF 691259582:691259582(0) win 1028
21:24:34.404488 166.104.219.69.21 > 46.5.154.192.21: SF 1366763947:1366763947(0) win 1028
21:25:59.534488 166.104.219.69.21 > 46.5.76.137.21: SF 205235563:205235563(0) win 1028
21:27:43.854488 62.153.209.202.21 > 46.5.243.229.21: SF 2089662048:2089662048(0) win 1028
21:40:04.534488 62.153.209.202.21 > 46.5.237.185.21: SF 376168805:376168805(0) win 1028
22:05:48.144488 166.104.219.69.21 > 46.5.164.57.21: SF 556242454:556242454(0) win 1028
22:13:49.934488 166.104.219.69.21 > 46.5.20.183.21: SF 1085078905:1085078905(0) win 1028
22:27:09.484488 62.153.209.202.21 > 46.5.171.123.21: SF 1286269942:1286269942(0) win 1028
22:40:41.904488 62.153.209.202.21 > 46.5.224.150.21: SF 987157143:987157143(0) win 1028
22:44:08.014488 166.104.219.69.21 > 46.5.166.59.21: SF 73357869:73357869(0) win 1028
22:52:01.514488 166.104.219.69.21 > 46.5.120.62.21: SF 876454982:876454982(0) win 1028
23:04:56.424488 166.104.219.69.21 > 46.5.56.10.21: SF 326236992:326236992(0) win 1028
23:11:59.694488 62.153.209.202.21 > 46.5.229.73.21: SF 1074458184:1074458184(0) win 1028
23:14:17.634488 62.153.209.202.21 > 46.5.26.199.21: SF 2053276638:2053276638(0) win 1028
23:26:57.344488 166.104.219.69.21 > 46.5.243.111.21: SF 1227832124:1227832124(0) win 1028
00:12:33.194488 166.104.219.69.21 > 46.5.67.194.21: SF 1287949398:1287949398(0) win 1028
00:30:22.784488 62.153.209.202.21 > 46.5.250.22.21: SF 996978313:996978313(0) win 1028
01:04:09.124488 166.104.219.69.21 > 46.5.214.19.21: SF 1381920016:1381920016(0) win 1028
01:38:59.144488 166.104.219.69.21 > 46.5.248.42.21: SF 337380566:337380566(0) win 1028
01:58:34.494488 166.104.219.69.21 > 46.5.28.67.21: SF 247125014:247125014(0) win 1028
02:08:40.014488 166.104.219.69.21 > 46.5.207.245.21: SF 1997306502:1997306502(0) win 1028
02:15:17.824488 62.153.209.202.21 > 46.5.44.48.21: SF 688616802:688616802(0) win 1028
02:46:21.674488 166.104.219.69.21 > 46.5.200.198.21: SF 1258321957:1258321957(0) win 1028
02:50:21.384488 166.104.219.69.21 > 46.5.41.6.21: SF 199101740:199101740(0) win 1028
02:53:25.284488 62.153.209.202.21 > 46.5.207.116.21: SF 2001006285:2001006285(0) win 1028
03:03:13.924488 166.104.219.69.21 > 46.5.156.246.21: SF 260041809:260041809(0) win 1028
03:12:11.114488 166.104.219.69.21 > 46.5.93.89.21: SF 158593050:158593050(0) win 1028
03:37:59.134488 62.153.209.202.21 > 46.5.122.90.21: SF 1590975133:1590975133(0) win 1028
03:50:12.484488 166.104.219.69.21 > 46.5.136.4.21: SF 847938763:847938763(0) win 1028
03:58:45.224488 166.104.219.69.21 > 46.5.224.110.21: SF 1592794866:1592794866(0) win 1028
04:06:40.554488 62.153.209.202.21 > 46.5.250.3.21: SF 1873322463:1873322463(0) win 1028
04:08:17.064488 62.153.209.202.21 > 46.5.139.196.21: SF 1662917236:1662917236(0) win 1028
04:15:31.174488 166.104.219.69.21 > 46.5.26.86.21: SF 586657448:586657448(0) win 1028
04:23:18.694488 166.104.219.69.21 > 46.5.199.141.21: SF 305338718:305338718(0) win 1028
04:48:41.054488 62.153.209.202.21 > 46.5.128.225.21: SF 1041097638:1041097638(0) win 1028
04:55:45.124488 166.104.219.69.21 > 46.5.249.227.21: SF 859883662:859883662(0) win 1028
```

```
04:59:42.344488 62.153.209.202.21 > 46.5.15.243.21: SF 505274227:505274227(0) win 1028
05:17:42.624488 62.153.209.202.21 > 46.5.191.170.21: SF 1685916193:1685916193(0) win 1028
05:25:44.114488 62.153.209.202.21 > 46.5.15.138.21: SF 1120959283:1120959283(0) win 1028
05:30:59.414488 166.104.219.69.21 > 46.5.10.87.21: SF 1115016828:1115016828(0) win 1028
05:31:58.524488 166.104.219.69.21 > 46.5.251.132.21: SF 47765149:47765149(0) win 1028
05:36:18.034488 62.153.209.202.21 > 46.5.102.4.21: SF 1585840476:1585840476(0) win 1028
06:02:38.484488 166.104.219.69.21 > 46.5.71.86.21: SF 938275777:938275777(0) win 1028
06:12:49.854488 166.104.219.69.21 > 46.5.129.154.21: SF 1960209197:1960209197(0) win 1028
06:39:25.164488 166.104.219.69.21 > 46.5.148.199.21: SF 1898552481:1898552481(0) win 1028
06:51:01.624488 166.104.219.69.21 > 46.5.196.82.21: SF 1760903014:1760903014(0) win 1028
06:58:58.894488 62.153.209.202.21 > 46.5.48.94.21: SF 1421392156:1421392156(0) win 1028
07:01:10.314488 166.104.219.69.21 > 46.5.4.147.21: SF 1226086601:1226086601(0) win 1028
07:03:18.464488 166.104.219.69.21 > 46.5.253.67.21: SF 2034481295:2034481295(0) win 1028
07:17:06.914488 166.104.219.69.21 > 46.5.247.174.21: SF 1154841220:1154841220(0) win 1028
07:19:26.674488 62.153.209.202.21 > 46.5.57.121.21: SF 537940161:537940161(0) win 1028
07:56:09.914488 62.153.209.202.21 > 46.5.169.200.21: SF 1628301232:1628301232(0) win 1028
08:00:35.834488 62.153.209.202.21 > 46.5.210.105.21: SF 1256745658:1256745658(0) win 1028
08:00:41.074488 62.153.209.202.21 > 46.5.228.147.21: SF 1892449285:1892449285(0) win 1028
08:14:13.154488 166.104.219.69.21 > 46.5.190.155.21: SF 747484591:747484591(0) win 1028
08:17:10.724488 166.104.219.69.21 > 46.5.3.230.21: SF 301280300:301280300(0) win 1028
08:23:52.794488 166.104.219.69.21 > 46.5.79.78.21: SF 1517711981:1517711981(0) win 1028
08:29:04.964488 166.104.219.69.21 > 46.5.242.164.21: SF 1118517137:1118517137(0) win 1028
08:58:40.254488 166.104.219.69.21 > 46.5.228.153.21: SF 11776125:11776125(0) win 1028
09:01:24.454488 62.153.209.202.21 > 46.5.102.42.21: SF 1742420710:1742420710(0) win 1028
09:41:46.414488 166.104.219.69.21 > 46.5.197.209.21: SF 596545340:596545340(0) win 1028
10:01:28.114488 62.153.209.202.21 > 46.5.172.252.21: SF 1970538656:1970538656(0) win 1028
10:01:42.454488 62.153.209.202.21 > 46.5.149.101.21: SF 2002884085:2002884085(0) win 1028
10:04:35.724488 62.153.209.202.21 > 46.5.72.210.21: SF 2005237737:2005237737(0) win 1028
10:21:19.244488 166.104.219.69.21 > 46.5.35.108.21: SF 1527696641:1527696641(0) win 1028
10:29:54.594488 62.153.209.202.21 > 46.5.17.133.21: SF 668278602:668278602(0) win 1028
10:47:08.834488 62.153.209.202.21 > 46.5.102.118.21: SF 504355440:504355440(0) win 1028
10:48:27.004488 62.153.209.202.21 > 46.5.206.25.21: SF 1482768662:1482768662(0) win 1028
10:55:29.584488 62.153.209.202.21 > 46.5.202.96.21: SF 1996484486:1996484486(0) win 1028
11:08:11.694488 62.153.209.202.21 > 46.5.166.52.21: SF 14901222:14901222(0) win 1028
11:10:31.284488 62.153.209.202.21 > 46.5.239.71.21: SF 1462707101:1462707101(0) win 1028
11:25:30.544488 62.153.209.202.21 > 46.5.131.13.21: SF 781406860:781406860(0) win 1028
12:04:48.844488 62.153.209.202.21 > 46.5.41.56.21: SF 606427278:606427278(0) win 1028
12:28:05.884488 62.153.209.202.21 > 46.5.62.86.21: SF 186554126:186554126(0) win 1028
12:46:57.984488 62.153.209.202.21 > 46.5.236.183.21: SF 265436364:265436364(0) win 1028
13:17:19.944488 62.153.209.202.21 > 46.5.12.135.21: SF 1272403862:1272403862(0) win 1028
14:15:08.834488 62.153.209.202.21 > 46.5.47.98.21: SF 675918527:675918527(0) win 1028
14:19:29.114488 62.153.209.202.21 > 46.5.175.117.21: SF 1826221884:1826221884(0) win 1028
14:26:20.344488 62.153.209.202.21 > 46.5.187.58.21: SF 303387681:303387681(0) win 1028
14:59:49.474488 62.153.209.202.21 > 46.5.72.99.21: SF 278390571:278390571(0) win 1028
15:05:03.174488 62.153.209.202.21 > 46.5.47.237.21: SF 348709009:348709009(0) win 1028
15:18:06.144488 62.153.209.202.21 > 46.5.128.80.21: SF 583364927:583364927(0) win 1028
15:23:35.474488 62.153.209.202.21 > 46.5.223.60.21: SF 384140395:384140395(0) win 1028
15:55:59.904488 62.153.209.202.21 > 46.5.13.179.21: SF 1572739694:1572739694(0) win 1028
16:40:56.664488 62.153.209.202.21 > 46.5.124.130.21: SF 1671830546:1671830546(0) win 1028
```

Note that the times are fairly well spaced out with not much in the way of a clear timed pattern. The attacker appears to be looking for available ftp server all within the 46.5 network. There is no pattern to the destinations (i.e. they don't flow in any order, but instead seem to be scattered everywhere). There's nothing obvious about the sequence numbers, although I think it is significant that the window size is 1028 consistently.

I was curious if I could determine anything about the attacking host, so I referenced the p0f fingerprint database (http://www.stearns.org/p0f/devel/p0f.fp) but did not find any instance of window size 1028. This causes me to think this was some sort of automated tool that did this.

Doing some searches on the Internet, I came across an article by Karen Kent Frederick that mentions the synscan tool is capable of generating this kind of traffic, and that it is sometimes related to the Ramen Worm (see http://online.securityfocus.com/infocus/1534 for details). Also in the article, it mentions that the IP Identification number should be 39426, and mentions that the reflexive port could either be 53 or 21. Let's look at a packet.

```
08:17:10.724488 166.104.219.69.21 > 46.5.3.230.21: SF [bad tcp cksum faf7!]
301280300:301280300(0) win 1028 (ttl 19, id 39426, len 40, bad cksum 623a!)
```

So there it is, id 39426 with source port 21. I'm pretty sure at this point I'm looking at synscan.

Running another filter of the logs, this time looking for the two attacking IP's 166.104.219.69 and 62.153.209.202 with any flag set, shows that the only traffic is from these attacker and two the above hosts with SF flags. No ACKS are seen, and no legitimate attempts at connecting to any other target are shown.

Attack mechanism

Because of the characteristics of the packet and its correlation with synscan, I believe a synscan engine was used to generate this traffic.

Correlations.

Proof for the synscan engine conclusion was mentioned above. Roland Gerlach (from http://www.giac.org/practical/Roland_Gerlach_GCIA.html#detect2) examined similar traffic and came to a similar conclusion about synscan, but noted that current versions of synscan use random IP ID's, so ones that still use 39246 are older variants.

Evidence of Active Targeting

None. There is no prior sweep, no indication of a DNS lookup, no evidence that the attacker is interested in any particular box, simply a sweep of the network looking for FTP.

Severity

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

Criticality: 5. I have no evidence as to what host is what, only that the entire network was scanned. Presumably, somewhere in this array of IP's, there's an important server.

Lethality: 1. This is a prelude to something else, the attack itself nothing more than a simple SF scan for port 21.

System Countermeasures: 5. No responses from any hosts indicate that there are no available FTP services running, at least none that are available from the attacker's point of origin.

Network Countermeasures: 5. Again, no response, while it could mean good system configuration, could also mean a strong firewall presence.

Severity: -4

Defensive Recommendations:

Verify no unnecessary FTP servers are running. If there are necessary FTP services running, attempt to lock down as much as possible, to include getting the most up to date patches and software from the vendor, setting up ACL's on the firewall to restrict who gets to it (if possible), and disable anonymous access to the FTP server (again, if possible). If possible, deploy firewall filters preventing inbound FTP connections.

Test Question

What is the purpose of someone scanning a network with both the SF flags set? Choose the best answer.

a. There is no purpose. This kind of scan won't work because its an invalid TCP flag set.
b. To evade IDS sensors.
c. Most machines when receiving a packet with both SF flags set will lock up, so this is more of a denial of service.

Answer: b

Part III - Analyze This

Executive Summary

Deploying a SNORT sensor on this network was a very wise action to take, but it is the tip of the iceberg as far as securing this network. As this is a University network, its primary function is as a tool of learning and research, as well as to provide staff and students a means of recreation. That having been said, one cannot expect a University network to be treated as a bank or government entity; the level of security achieved in either institution would be too restrictive to foster the kind of atmosphere the University is presumably trying to portray. A security administrator at a University has a very unique challenge: to balance the needs of the University with the netizen related responsibilities of preventing the spread of hostile code.

Judging by the type of scans being seen on the network, it is easy to see that all facets of network behavior are being exhibited. The good (legitimate Internet traffic), the bad (compromised hosts spreading worms), and the ugly (recreational users tying up tremendous amounts of bandwidth) all play a role in this network. If we approach this situation from the question, "Are we permitting our users every kind of access they could ask for?" the answer is a resounding yes. The University is doing a fantastic job of allowing any kind of traffic a network user could hope for. However, I have been tasked to approach from the question, "How secure is the network?" and the answer is, not at all.

There is tremendous evidence of compromised systems running rampant throughout the network. There are indicators of mischievous behavior, users taking advantage of the permissive network policies and tying up valuable bandwidth. It is evident that little has been done to secure the network, and while the University's mission might be to promote an open network, it needs to be understood that such policies can have a detriminental impact to University users.

A student whose machine has been overrun with worms and viruses likely will not be able to pursue academic objectives. The research of a University scientist who cannot access the latest research website because all bandwidth is being tied up with games and music file sharing will likely suffer. If the University's primary website or email server gets compromised, potentially sensitive data relating to students and staff could be transmitted to unauthorized users, resulting in law suits and liability, not to mention loss of reputation.

The recommendations I present throughout this report are designed to present system and academic administrators with a very basic roadmap to getting them on the path to a secure network. It may be that some suggestions will be discarded because they too severely limit the users from legitimate tasks; these are choices that every administrator makes regardless of the company they work for. But is my duty to ensure that the reader understands what is happening on the network, what is at stake, what are the options, and what are the consequences of those options.

List of Files Analyzed:

Obtained from http://www.incidents.org/logs.

alert.020801.txt
alert.020802.txt
alert.020803.txt
alert.020804.txt
alert.020805.txt

oos_Aug.1.2002.txt
oos_Aug.2.2002.txt
oos_Aug.3.2002.txt
oos_Aug.4.2002.txt
oos_Aug.5.2002.txt

scans.020801.txt
scans.020802.txt
scans.020803.txt
scans.020804.txt
scans.020805.txt

The alert files, as they are provided by giac.org, provide a wealth of information in a format not too easy to manage in huge volumes. Ron Clark, a fellow GCIA hopeful, came across a perl script by Tod Beardsley that takes snort log files generated in fast style and converts them to a comma deliminated format (see http://www.giac.org/practical/Tod_Beardsley_GCIA.doc for details). From there, they can be loaded into a MySQL database and queried using standard SQL syntax.

The problem, however, is that the log files as they were given were not in a consistent format. Normally, its one alert for every line, but in some cases, the alerts were given two alerts on one line. These had to be corrected, but doing so within 600+ MB of data is no easy task. I therefore wrote the following perl script to sort through the logs and correct any supposed mistakes.

```
#!/usr/bin/perl
#
# These 3 variables should be adjusted to fit the individual
# administrator's needs.

$dir = "/work/MA";
$input = "alert.020805.txt";
```

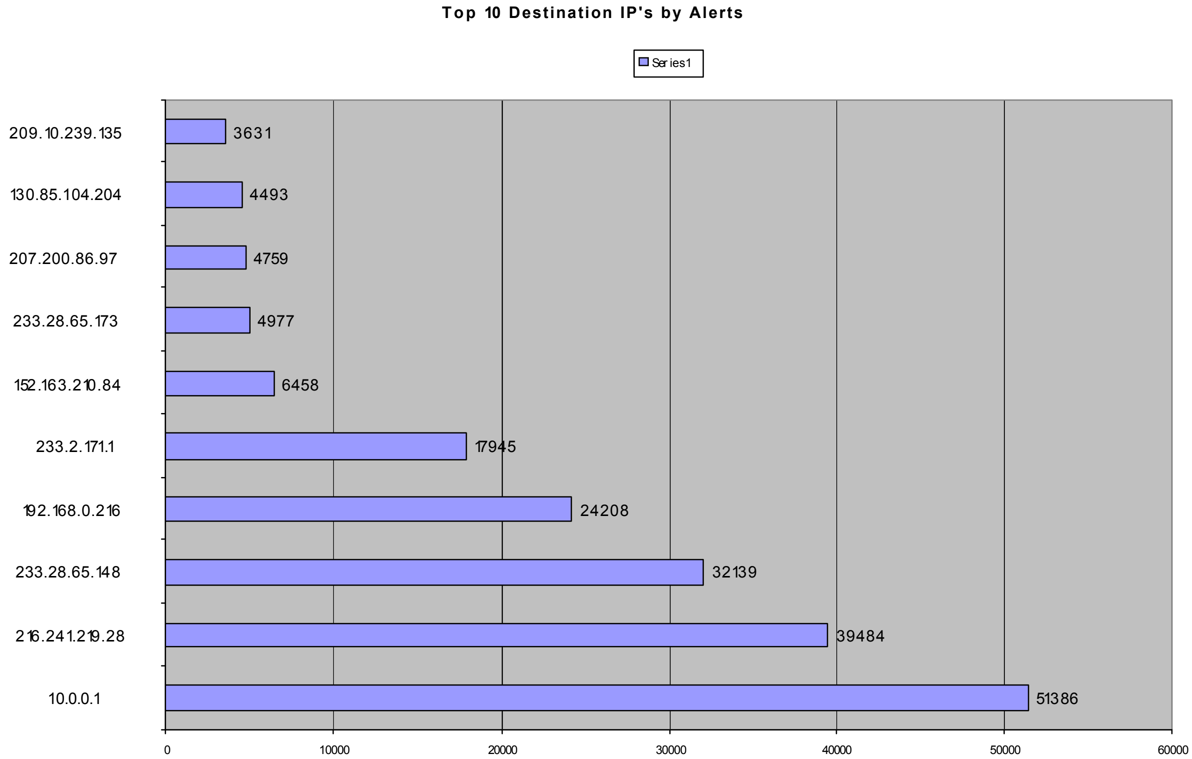
```
$output = "correct.020805.txt";

open(FILE,"$dir/$input");
@FILE_LINES = <FILE>;

open(CORRECT,">$dir/correct/$output");
$count = 1;
foreach $dataline (@FILE_LINES) {
        $dllength = length($dataline);
# The theory is that lines over 150 characters in length are candidates
# for having multiple alerts on the same line.  Any that are under 150,
# ignore.  Any that are over 150, locate any instances of "08/0" and
# replace with "\n08/0" thus causing a line break.  It also tells you what
# line number it located the error.
        if ($dllength >= 150) {
                $dataline =~ s/.08\/0/\n08\/0/g;
                print CORRECT ("$dataline");
                print ("$count, ");
        } else {
                print CORRECT ("$dataline");
        }
        $count++;
}

close FILE;
close CORRECT;
```

Once all the alert files were corrected, I then proceeded to comma deliminate them using Tod Beardsley's csv.pl script, create a MySQL database with appropriate tables and fields, and load them into the database using the following MySQL syntax.

```
LOAD DATA INFILE '/work/MA/correct/correct.020801.txt.csv' INTO TABLE
alerts FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
```

High Level Overview, Charts, and Graphs

I used MS Excel to make the following three charts. It shows Top 10 Source IP’s, Destination IP’s, and Top 10 Alerts over the 5 day period.



	10.0.0.1	216.241.219.28	233.28.65.148	192.168.0.216	233.2.171.1	152.163.210.84	233.28.65.173	207.200.86.97	130.85.104.204	209.10.239.135
■ Series1	51386	39484	32139	24208	17945	6458	4977	4759	4493	3631

Top 10 Destination IP's based on Alerts (Chart)

I'd like to single out the top talker here since it is actually a non-routable IP address, indicating possible automated scanning by worms within the network. It is unlikely anyone was able to send us a packet destined for the 10 network since it would had to have been routed at every hop between us and the attacker. Again, this definitely smells like we have some machines compromised. A list of alerts generated to 10.0.0.1 turns up...

```
mysql> select distinct(alert_name), count(*) as totalcount from alerts
where dest_ip = '10.0.0.1' group by alert_name order by totalcount desc;
+----------------------------------------------------+------------+
| alert_name                                         | totalcount |
+----------------------------------------------------+------------+
| UDP SRC and DST outside network                    |      51374 |
| NIMDA - Attempt to execute cmd from campus host    |          4 |
| spp_http_decode: IIS Unicode attack detected       |          3 |
| IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize |       2 |
| SMB Name Wildcard                                  |          1 |
| Possible trojan server activity                    |          1 |
| NIMDA - Attempt to execute root from campus host   |          1 |
+----------------------------------------------------+------------+
```

UDP SRC and DST outside network would seem to indicate traffic that should not hit the snort sensor. Looking at all source IP's that tried talking to 10.0.0.1 and triggered that alert...

```
mysql> select distinct(source_ip), count(*) as totalcount from alerts where
dest_ip = '10.0.0.1' and alert_name = 'UDP SRC and DST outside network'
group by source_ip order by totalcount desc;
+---------------+------------+
| source_ip     | totalcount |
+---------------+------------+
| 3.0.0.99      |      51373 |
| 63.250.213.12 |          1 |
+---------------+------------+
```

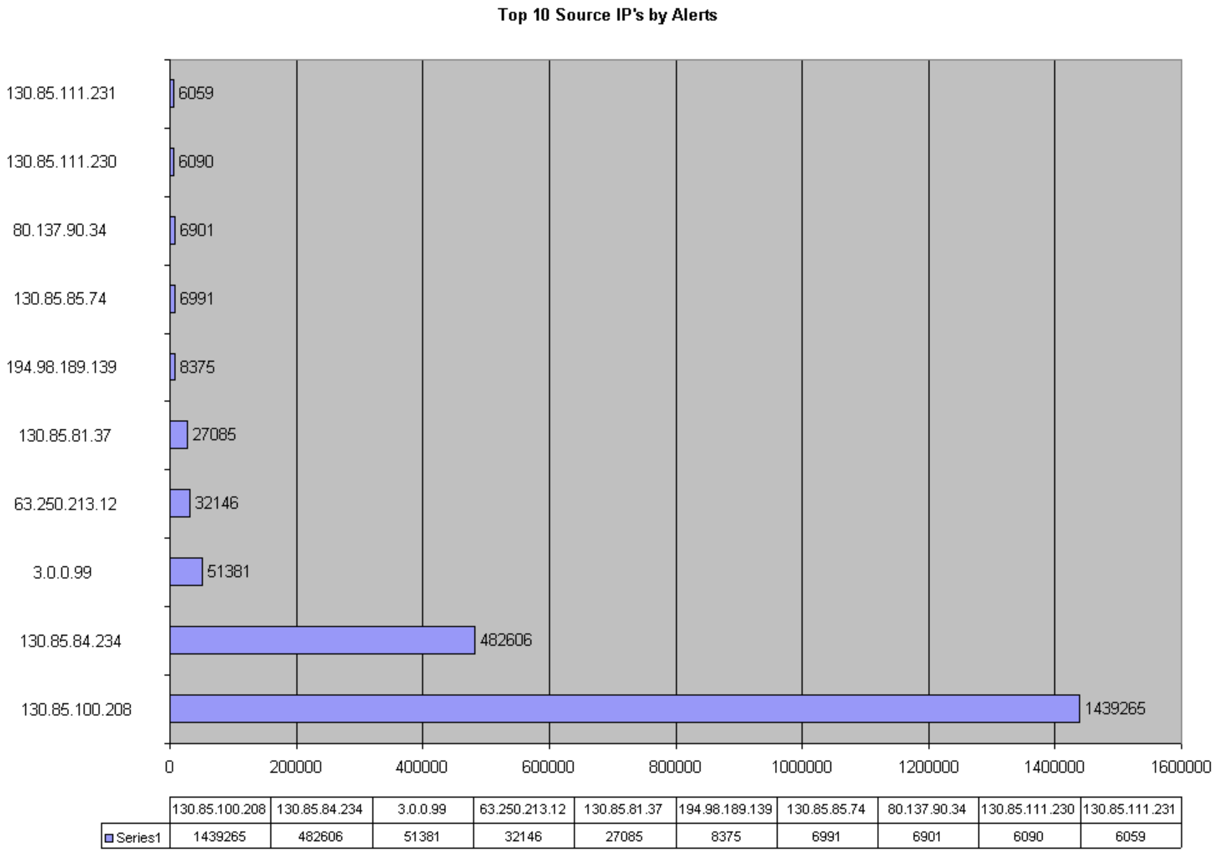
A sample alert that illustrates this...

```
08/02-03:32:16.372187  [**] UDP SRC and DST outside network [**]
3.0.0.99:137 -> 10.0.0.1:137
```

This at first glance this looks like a massive Windows netbios transmission. However, neither 3.0.0.99 or 10.0.0.1 are considered to be apart of the internal network, so how is our SNORT sensor picking this up?

Rick Yuen, in is GCIA practical (http://www.giac.org/practical/Rick_Yuen_GCIA.doc) indicated that there might be a misconfigured box with an improper WINS address, and so the box is sending query after query looking for the non-existent WINS server. I was unable to find any other correlations for this kind of network behavior.

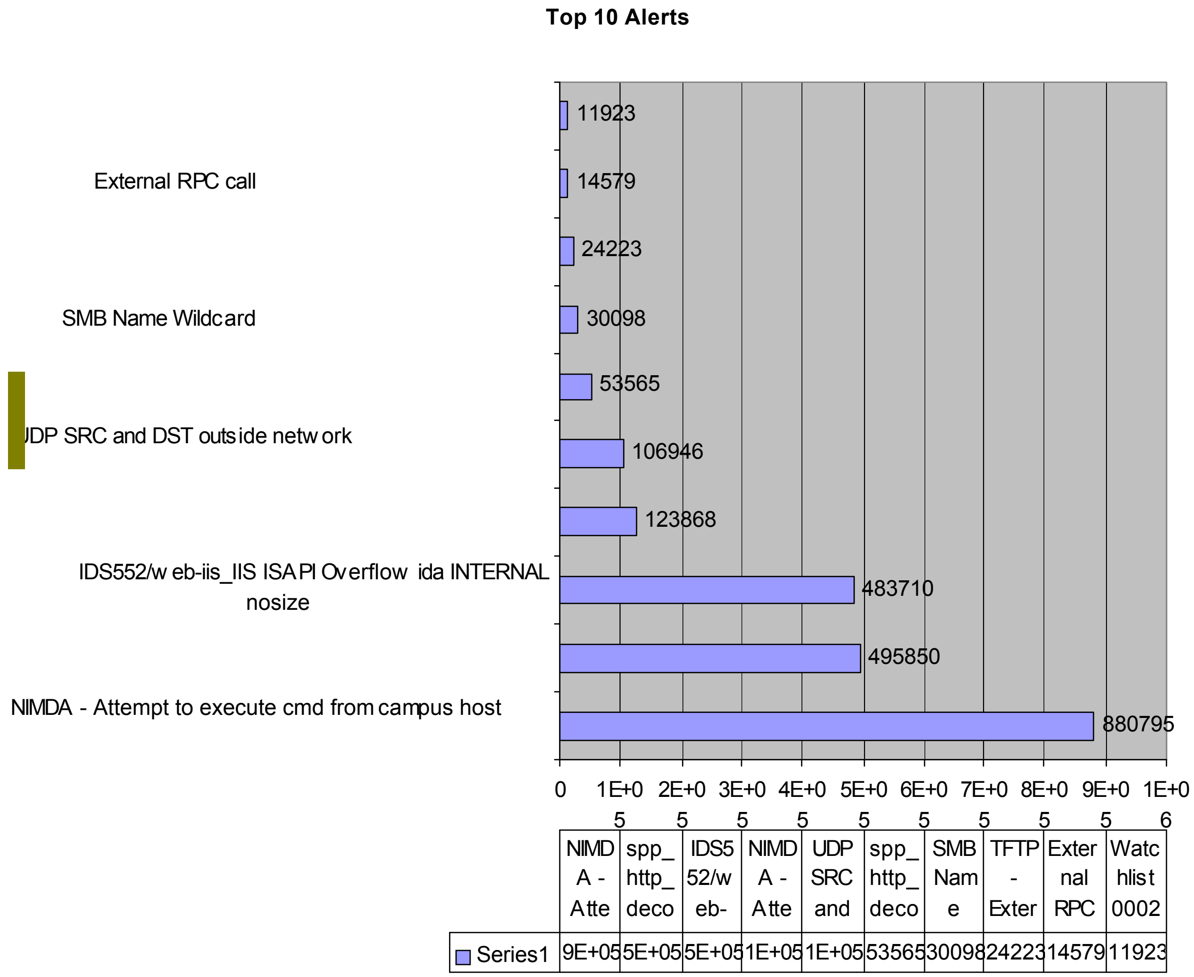
I flag this as unusual activity, and I feel it would be worth the administrator's time to go and investigate further using whatever resources they have at their disposal to get to the bottom of this.

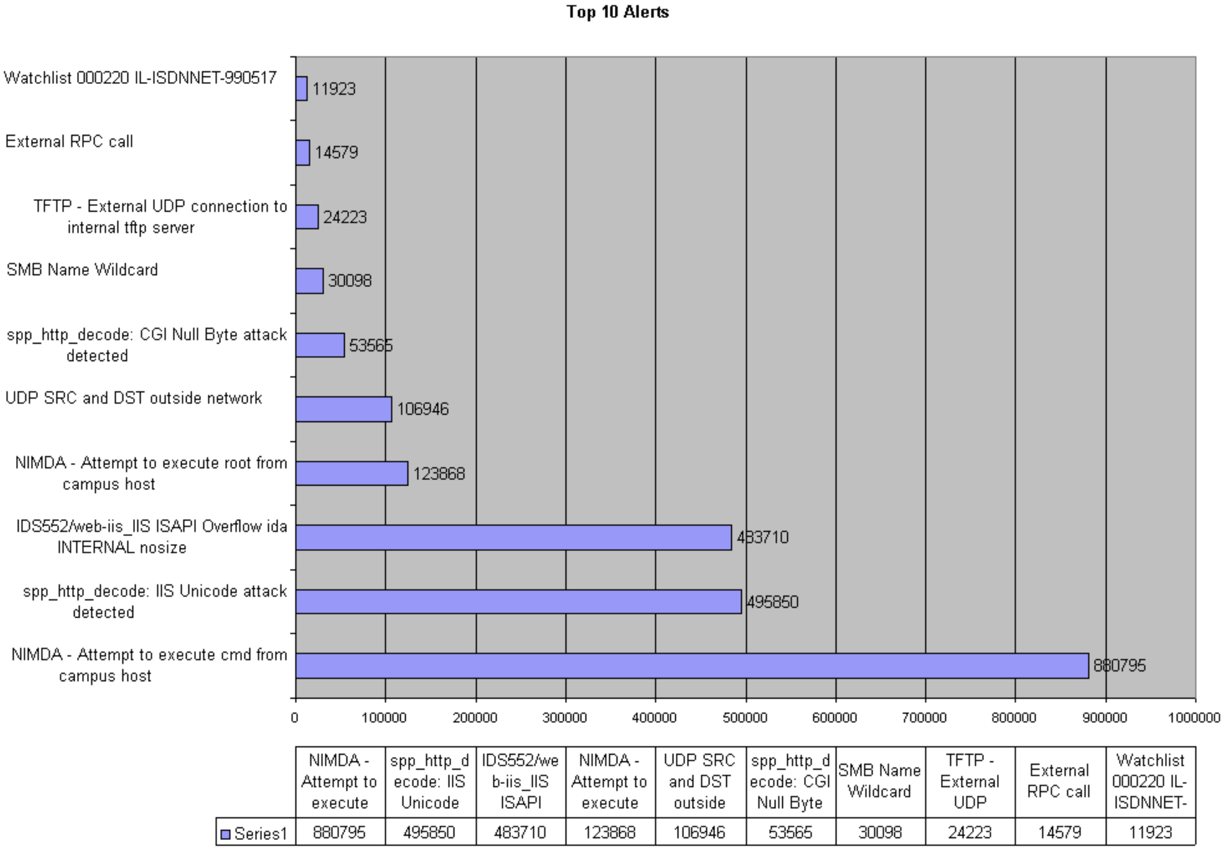


	130.85.100.208	130.85.84.234	3.0.0.99	63.250.213.12	130.85.81.37	194.98.189.139	130.85.85.74	80.137.90.34	130.85.111.230	130.85.111.231
▨ Series1	1439265	482606	51381	32146	27085	8375	6991	6901	6090	6059

Top 10 Source IP's based on Alerts (Chart)

It's interesting to note here that the top two sources of alerts, my.net.100.208 and my.net.84.234 are actually members of our home network. While this shows top 10 source IP's based on number of triggered alerts, it will be worthwhile to go back and generate a list of home addresses that supposedly originated some of these alerts. I will do this further in the paper.

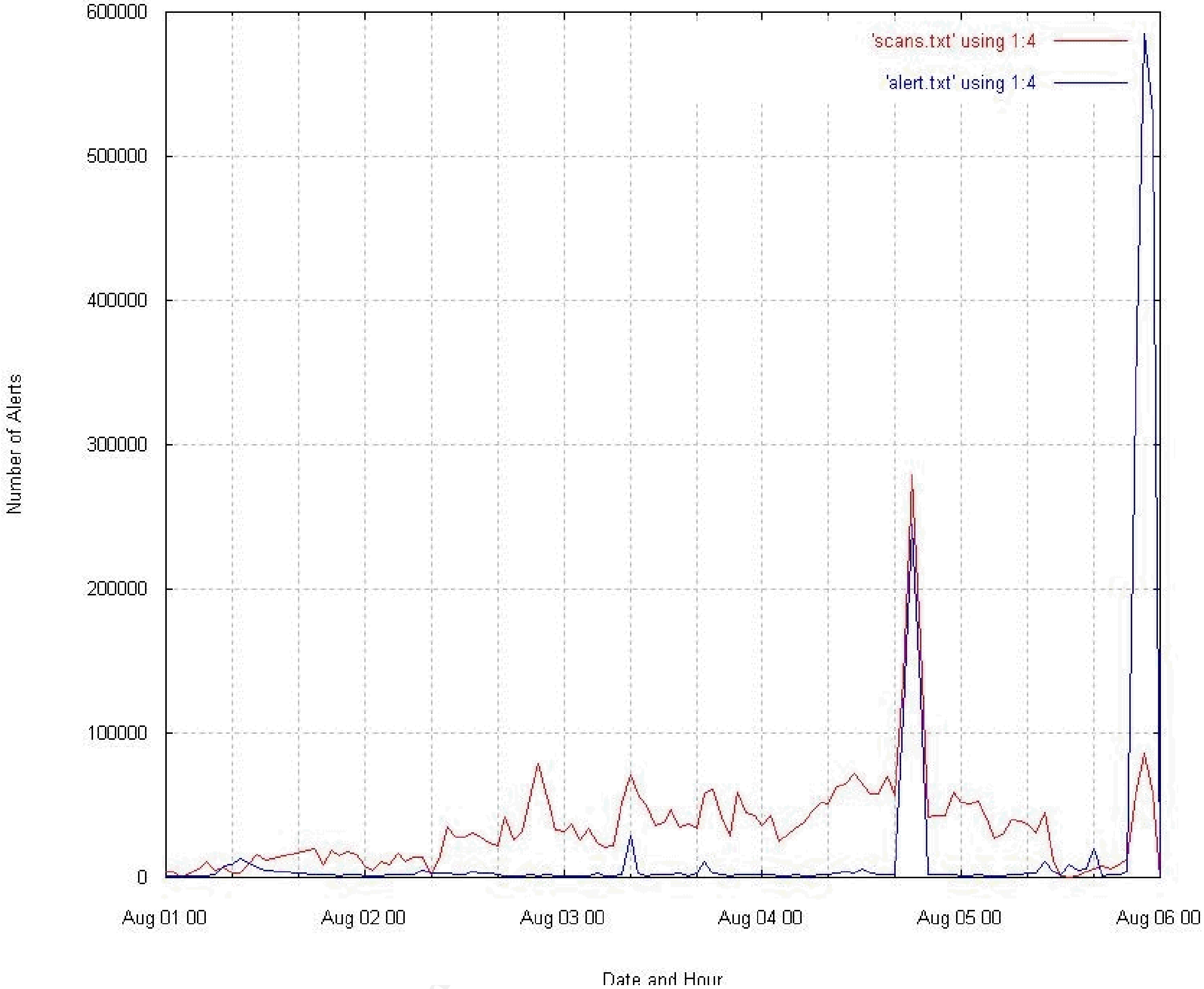

Top 10 Alerts
External RPC call
SMB Name Wildcard
JDP SRC and DST outside network
IDS552/w eb-iis_IIS ISAPI Overflow ida INTERNAL nosize
NIMDA - Attempt to execute cmd from campus host
11923
14579
24223
30098
53565
106946
123868
483710
495850
880795
0
1E+05
2E+05
3E+05
4E+05
5E+05
6E+05
7E+05
8E+05
9E+05
1E+06
NIMDA - Atte
spp_http_deco
IDS552/web-
NIMDA - Atte
UDP SRC and
spp_http_deco
SMB Name
TFTP - Exter
External RPC
Watchlist 0002
Series1
9E+05
5E+05
5E+05
1E+05
1E+05
53565
30098
24223
14579
11923



	NIMDA - Attempt to execute	spp_http_decode: IIS Unicode	IDS552/web-iis_IIS ISAPI	NIMDA - Attempt to execute	UDP SRC and DST outside	spp_http_decode: CGI Null Byte	SMB Name Wildcard	TFTP - External UDP	External RPC call	Watchlist 000220 IL-ISDNNET-
■ Series1	880795	495850	483710	123868	106946	53565	30098	24223	14579	11923

Top 10 Alerts (Chart)

This chart was especially useful because the very top alert is a Nimda attempt from within the campus, definitely signs that we've been compromised.



Count of Scans and Alerts plotted by Time (Graph)

I used Gnuplot (http://www.gnuplot.info/) to help chart the scans and alerts over a five day period. It shows number of alerts and scans vs. day and hour. A few spikes would indicate possible problem areas that may need examination, specifically Aug 4 between 17:00 and 19:00, and also Aug 5 between 22:00 and 23:59. Note that the scans and alerts peek at similar time intersections. This could indicate that many snort rules are set up that overlap the portscan preprocessor, which is usually a sign of too general a snort rule.

On Aug 4 between 17:00 and 19:00, we see a large number of ISAPI overflow alerts originating from my.net.84.234 directed towards a vast array of external addresses. This correlates to a large number of port 80 scans from my.net.84.234.

On Aug 5 between 22:00 and 23:59, we see a large number of port 80 scans originating from an internal host, my.net.100.208. This could indicate a possible compromise of that system. This correlates to many IIS Unicode and Nimda type attacks directed at that host.

Most Severe Alerts

Methodology

In determining what I categorized as most severe, I first needed to determine what network I was looking at. First, I generated a list of all distinct alert names and flagged the ones that made reference to "internal" or "external". Hopefully, these rules would establish a pattern for subnets that the IDS snort admin is concentrating on, and tell me what his $HOME network is. Based on the pattern here, I ran a query against the alerts database looking only for alerts where a home network IP was the source. This was the result.

```
select distinct(alert_name), count(*) as totalcount from alerts where
source_ip LIKE 'my.net%' group by alert_name order by totalcount desc;
```

```
+--------------------------------------------------------------+------------+
| alert_name                                                   | totalcount |
+--------------------------------------------------------------+------------+
| NIMDA - Attempt to execute cmd from campus host              |     880795 |
| IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize        |     483710 |
| spp_http_decode: IIS Unicode attack detected                 |     481844 |
| NIMDA - Attempt to execute root from campus host             |     123868 |
| spp_http_decode: CGI Null Byte attack detected               |      53531 |
| TFTP - External UDP connection to internal tftp server       |      24218 |
| IRC evil - running XDCC                                      |       2055 |
| Possible trojan server activity                              |       1807 |
| High port 65535 udp - possible Red Worm - traffic            |        247 |
| TFTP - Internal UDP connection to external tftp server       |        166 |
| beetle.ucs                                                   |         83 |
| Port 55850 tcp - Possible myserver activity - ref. 010313-1 |         77 |
| High port 65535 tcp - possible Red Worm - traffic            |         24 |
| HelpDesk my.net.70.50 to External FTP                        |         11 |
| HelpDesk my.net.70.49 to External FTP                        |          9 |
| HelpDesk my.net.83.197 to External FTP                       |          4 |
| TFTP - External TCP connection to internal tftp server       |          3 |
| Port 55850 udp - Possible myserver activity - ref. 010313-1 |          2 |
| Traffic from port 53 to port 123                             |          1 |
+--------------------------------------------------------------+------------+
```

This is the list of most severe alerts. Excluding external hosts from the data is useful because it reduces the clutter in diagnosing the severity of an event. Not that data originating from external sources is irrelevant, far from it. But often signs of misuse turn up by a compromised host trying to initiate communication outside the network. Correlating these outbound connections with alerts inbound to the suspect host provides the necessary confirmation that an internal host has, indeed, been compromised.

Further, without knowledge about the various hosts on the internal network and their OS, what services they run, their patch level, what is considered "expected" network behavior, etc., it would be difficult to gauge whether an attack had succeeded against the host without subsequent traffic coming from the host outbound.

I've taken the top 8 alerts by count and gauged the severity also by the type of alert generated. I gave preference to alerts that showed possible signs of internal host compromise. For example, I did not rate the "Traffic from port 53 to port 123" because there was only one count and does not immediately indicate signs of compromise.

After examining these 8 alerts, I then went on to the scans and out of spec files looking for out of the ordinary activity that wasn't caught by the alerts.

The 1st Most Severe Alert(s)

Nimda – Attempt to execute cmd from campus host
Nimda – Attempt to execute root from campus host
spp_http_decode: IIS Unicode attack detected
TFTP - Internal UDP connection to external tftp server

I've lumped these alerts all in one section because I feel that they are all interconnected and indicative of a single worm: Nimda. It's is a fairly well document worm that propagates through a number of different methods. It injects itself deep into the host's Windows operating system and attempts to generate emails destined for recently used addresses and then send them out using its own direct connect to smtp sockets. Further it launches probes looking for other machines that are listening over port 80, and then executes a unicode folder traversal against that host in order to spread the worm. It opens up a tftp service local on the infected server so that it can upload key infected files to future victims. It looks for open netbios file shares in order to spread the worm and, as if this weren't enough, it copies an infected email into the websites default virtual directories so even unsuspecting Internet users visiting the webpage might get the worm. See http://rr.sans.org/malicious/nimda3.php for more details on this worm.

Executing cmd or root is a polite way of saying that they're attempting to get into other web hosts. By running specially formatted commands in the GET request, a person could attempt to enumerate the web hosts c: drive, or create new text files using the "type" function (such as a new default.htm). Root.exe is commonly created by viruses, such as Code Red (see http://www.cert.org/advisories/CA-2001-13.html for details on Code Red) by copying the cmd.exe from a Windows system's winnt\system32 directory to the /scripts virtual web directory, renaming it to root.exe in the process. Thus, running root.exe would accomplish the same thing as running cmd.exe.

The IIS Unicode attack makes reference to a flaw in the way IIS handles unicode packets. Unicode is a kind of hex to ascii conversion you can do to represent common ascii characters in non-ascii format. Typically these are represented by the "%" symbol, followed immediately by the two hex digit representation of an ascii character. IIS passes the unicode directly to the application before attempting to decode the meaning of those symbols. Thus, controls put in place that keep web users from going to places they shouldn’t be going are bypassed because the requests are sent in unicode format.

A list of all source IP's that generated Nimda style alerts...

```
mysql> select distinct(source_ip), count(*) as totalcount from alerts where alert_name LIKE
'NIMDA%' group by source_ip order by totalcount desc;
+----------------+------------+
| source_ip      | totalcount |
+----------------+------------+
| my.net.100.208 |    1001167 |
| my.net.100.20  |       3487 |
| my.net.82.87   |          1 |
| my.net.70.16   |          1 |
| my.net.83.176  |          1 |
| my.net.111.30  |          1 |
| my.net.165.19  |          1 |
| my.net.70.169  |          1 |
| my.net.70.144  |          1 |
| my.net.105.10  |          1 |
| my.net.130.20  |          1 |
+----------------+------------+
```

Well, it seems pretty bad for my.net.100.208 and my.net.100.20. Very likely, these are infected with the Nimda virus. Curious as to why it shows 9 other IP’s in the same subnet that exhibited trace amounts of NIMDA like activity. This should be worth the administrator’s time in going to each of these machines and determining what’s going on. It’s also my humble opinion that sorting for alerts from my.net.100.208 is worth an extra look. Thus…

```
mysql> select distinct(alert_name), count(*) as totalcount from alerts
where source_ip LIKE 'my.net.100.208' group by alert_name order by
totalcount desc;
+---------------------------------------------------------+------------+
| alert_name                                              | totalcount |
+---------------------------------------------------------+------------+
| NIMDA - Attempt to execute cmd from campus host         |     877731 |
| spp_http_decode: IIS Unicode attack detected            |     437935 |
| NIMDA - Attempt to execute root from campus host        |     123436 |
| TFTP - Internal UDP connection to external tftp server  |        163 |
+---------------------------------------------------------+------------+
```

Looking at the other alert, ' TFTP - Internal UDP connection to external tftp server' for unique destinations, it turns up...

```
mysql> select distinct(dest_ip), count(*) as totalcount from alerts where
alert_name = 'TFTP - Internal UDP connection to external tftp server' group
by dest_ip order by totalcount desc;
+----------------+------------+
| dest_ip        | totalcount |
+----------------+------------+
| 192.168.0.1    |         64 |
| 169.254.97.57  |         45 |
| 61.145.69.74   |         25 |
| 169.254.126.97 |         19 |
| 211.141.120.18 |         10 |
| my.net.180.39  |          3 |
| 130.75.1.144   |          3 |
| my.net.178.76  |          2 |
| my.net.117.25  |          1 |
| my.net.84.189  |          1 |
+----------------+------------+
```

Again, 192.168.0.1, but this time we have 169.254.97.57, 61.145.69.74, 169.254.126.97, and 211.141.120.18. Let's look at each of these.

my.net.100.208 sent 45 TFTP packets to 169.254.97.57 and 19 to 169.254.126.97. No response packets were logged. Unclear whether this was a successful TFTP transfer of if it was just repeated attempts. No other alerts are flagged from 169.254.97.57 or 169.254.126.97.

61.145.69.74 is a different story, however. Not only do we see my.net.100.208 attempting a TFTP transfer, but 61.145.69.74 made numerous IIS Unicode attacks against it as well. This looks very much like 61.145.69.74 was responsible for the resultant TFTP'ing by my.net.100.208.

211.141.120.18 looks similar, but very subtly different. While we see IIS Unicode attacks coming from this host to my.net.100.208, we don't see them until after my.net.100.208 makes the TFTP connection. 61.145.69.74 does the IIS Unicode attack first, then we see the TFTP.

This box is definitely hacked and is trying to spread the disease. We see not only it trying to propagate to other unsuspecting servers, but also attempting tftp transfers as per standard Nimda behavior.

Correlations:

The behavior seen from these detects is similar to the original CERT advisory that was posted addressing the Nimda worm. Reference http://www.cert.org/advisories/CA-2001-26.html.

Defensive Recommendations:

Scrub my.net.100.208 and my.net.100.20 for Nimda per http://rr.sans.org/malicious/nimda2.php. Recommend also applying the Microsoft patch that address the vulnerabilities Nimda exploits. See http://www.microsoft.com/technet/security/bulletin/MS01-044.asp for details on patching.

So we accounted for 163 instances of TFTP, but we had over 24000 alerts for TFTP in general. So who’s doing all the tftp talking? This leads us to the second most severe alerts…

2nd Most Severe

Alert(s):
TFTP - External UDP connection to internal tftp server.

The following is a list of machines that exhibited some instances of TFTP behavior.

```
mysql> select distinct(source_ip), count(*) as totalcount from
alerts where alert_name LIKE 'TFTP%' group by source_ip order by
totalcount desc;
+-----------------+------------+
| source_ip       | totalcount |
+-----------------+------------+
| my.net.111.230  |       6090 |
| my.net.111.231  |       6059 |
| my.net.109.105  |       6054 |
| my.net.111.219  |       6007 |
| my.net.100.208  |        163 |
| my.net.111.23   |          5 |
| 209.61.187.112  |          5 |
| my.net.114.45   |          3 |
| 24.209.230.143  |          3 |
| my.net.82.130   |          3 |
| my.net.109.10   |          2 |
| 195.92.252.254  |          2 |
| 199.106.211.166 |          2 |
| 12.129.73.230   |          1 |
| 64.38.251.140   |          1 |
| 63.250.205.12   |          1 |
| my.net.111.21   |          1 |
+-----------------+------------+
```

So we've got my.net.111.230, my.net.111.231, my.net.109.105, and my.net.111.219 that are exhibiting high amounts of TFTP. Our unseen IDS admin is presumably logging this traffic via SNORT because it is out of spec, and so I assume this isn't supposed to be happening.

I run another SQL query looking for who the source of these 'TFTP - External UDP connection to internal tftp server' alerts is.

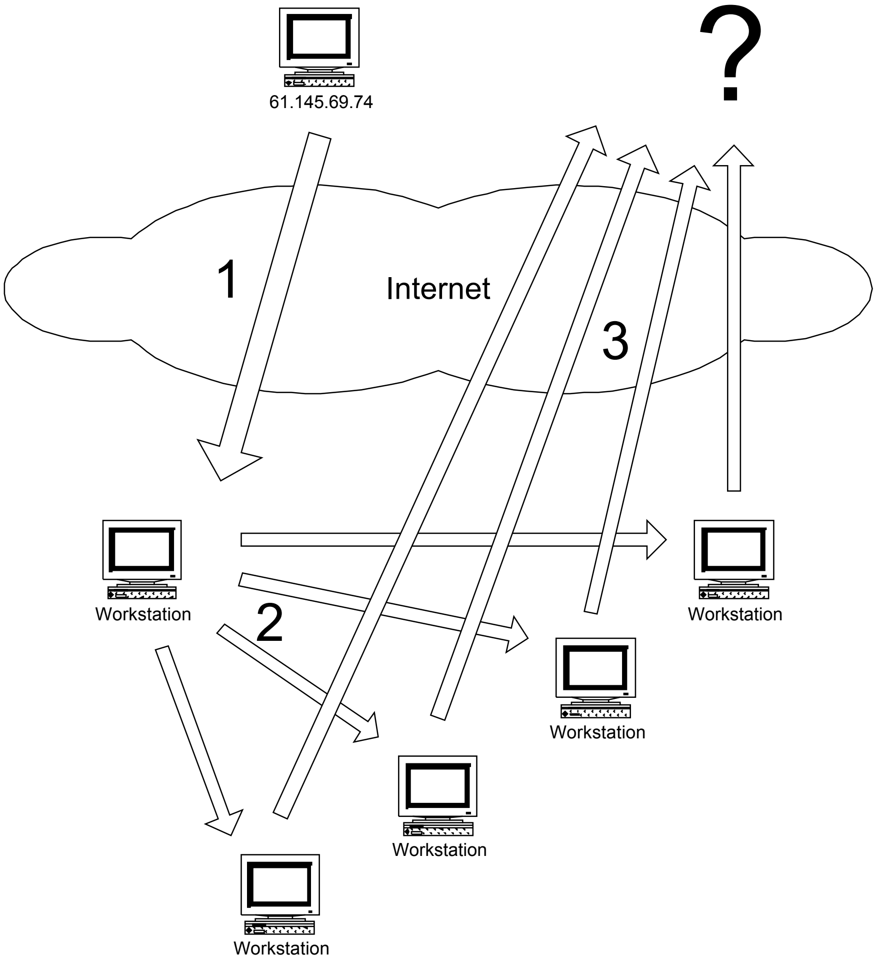
```
mysql> select distinct(dest_ip), count(*) as totalcount from alerts where
alert_name = 'TFTP - External UDP connection to internal tftp server' group
by dest_ip order by totalcount desc;
+---------------+------------+
| dest_ip       | totalcount |
+---------------+------------+
| 192.168.0.216  |      24207 |
|                |           8 |
| 192.168.0.21   |           2 |
| my.net.180.39  |           2 |
| my.net.117.25  |           2 |
| my.net.104.204 |           1 |
| my.net.114.44  |           1 |
+----------------+------------+
```

Indicates that the vast majority of these alerts is destined from a non-routable IP subnet, 192.168. Investigating further shows that the four IP's exhibiting the 6000+ count of TFTP are really going after 192.168.0.216. This could indicate any number of possibilities, either the server is compromised, there's a misconfiguration on the host causing it to want to connect to a TFTP server on an internal subnet, or these packets are entirely spoofed from somewhere within the network. Either way, its suspicious and warrants further investigation outside of looking at these alert logs.

Now I'm worried about damage control. It's pretty clear that my.net.100.208 was hacked, but I want to see if its exhibiting any other symptoms of compromise. I run another query only looking for alerts from my.net.111.230, my.net.111.231, my.net.109.105, and my.net.111.219, but it doesn’t turn up any other alerts. So I run another query, this time looking for all alerts destined to those four IP addresses. Each of the four turn up only two alerts…

External RPC call
SMB Name Wildcard

If what we're looking for is Nimda type alerts, I don't see it. Nimda does spread by way of mapped network drives, which could explain the SMB Name Wildcard, but I see no web attack, and External RPC calls does not fit the Nimda schema. It looks like what may have happened is that my.net.100.208 was the server originally infected and then it spread Nimda within the network, something our snort sensor probably wouldn't pick up. The following diagram illustrates this theory. I don't think there is enough evidence to say one way or the other whether this is happening or not, just that it's plausible.



Data Link Diagram

1. 61.145.69.74 launches a Nimda attack against a single internal host, in this case, my.net.100.208. This host becomes infected and begins TFTP'ing back to the original attacker.

2. my.net.100.208 becomes infected and proceeds to infect other internal servers. Our sensor won't pick this up because its only watching traffic going to or from the Internet.

3. We see various internal hosts now infected with Nimda and trying to TFTP to a non-routable address 192.168.0.216.

Just to be thorough, I want to know who is initiating these RPC calls and if they were involved with any other activity to anyone else. Two attacker hosts popped on this query: 194.98.189.139 and 61.182.50.241, and each of these were all over the IP map, definitely looks like they were openly scanning for RPC traffic as well as blindly spraying STATDX into the mix. Another query looking for any alert sourced from either of these two attackers shows…

61.182.50.241

```
+-------------------+------------+
| alert_name        | totalcount |
+-------------------+------------+
| External RPC call |       4519 |
| STATDX UDP attack |         10 |
+-------------------+------------+
```

194.98.189.139

```
+-------------------+------------+
| alert_name        | totalcount |
+-------------------+------------+
| External RPC call |       8352 |
| STATDX UDP attack |         23 |
+-------------------+------------+
```

While STATDX UDP is referencing an RPC buffer overflow attempt, details of which can be found at http://online.securityfocus.com/bid/1480/discussion/, neither of these two attacking hosts hit any of the 4 TFTP'ing hosts. No correlation between the RPC and TFTP that I can tell.

Defensive Recommendations:

It is highly recommended that the IDS analyst visit my.net.111.230, my.net.111.231, my.net.109.105, and my.net.111.219 for possible Nimda infection, and patch the servers accordingly. If these hosts are not infected, determine the cause of origin for the outbound TFTP connects by looking for any sign of compromise. Also determine if web services needs to be running on these hosts.

Egress filtering on the firewall, or blocking outbound connections from inside the network, is highly advisable as it would prevent the kind of traffic we're seeing.

3rd Most Severe

Alert

spp_http_decode: CGI Null Byte attack detected

This one was a lot of fun to research. The CGI Null Byte attack, also known as a poison null byte attack, is well documented by a fairly well known security-conscious programmer, Rain Forest Puppy. It’s premise is that most cgi scripts written in perl make certain assumptions about null bytes, or %00. The idea is that “string” is not the same as “string%00” in perl. However, in other program languages, tacking on the null byte yields the same result as when it isn’t there. If someone pulls up a perl cgi script and starts putting in null byte values into various text fields, its usually a hint that they are attempting to circumvent some sort of built in controls within the cgi. Depending on the nature of the script and how its being exploited, it could give the attacker much more visibility and access than is desired. See http://www.wiretrip.net/rfp/p/doc.asp/i2/d6.htm for more details.

The upshot is that there are many prepackaged perl scripts out there that are vulnerable to this kind of attack.

In addition to the cgi aspect of the vulnerability, there's another component involving Internet Explorer and how it handles file names/content types that contain null bytes. There are some files that IE will automatically download and run for the "convenience" to the end user. An example of this includes .txt files. However, .exe's would cause IE to prompt the user for whether they want to save or run the program in question. It is possible to insert a null byte into a file name that is really an exe but make IE think its a txt, thus causing it to automatically be pulled and run. MS01-058 has more details on this as well as patch information which is located at http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-058.asp.

A list of the source IP’s exhibiting this behavior…

```
+----------------+------------+
| source_ip      | totalcount |
+----------------+------------+
| my.net.81.37   |      27083 |
| my.net.182.91  |       5379 |
| my.net.178.219 |       5085 |
| my.net.109.83  |       4176 |
| my.net.87.52   |       3691 |
| my.net.70.48   |       3578 |
| my.net.84.189  |       1991 |
| my.net.85.78   |       1164 |
| my.net.107.192 |        300 |
| my.net.99.191  |        234 |
| my.net.88.155  |        217 |
| my.net.145.160 |        166 |
| my.net.87.103  |        121 |
| my.net.86.20   |         67 |
| my.net.178.78  |         60 |
| my.net.83.215  |         55 |
| my.net.104.49  |         40 |
| my.net.84.245  |         38 |
| my.net.104.143 |         22 |
| my.net.108.46  |         14 |
| my.net.153.206 |         10 |
| my.net.139.41  |          9 |
| 12.91.163.153  |          9 |
| 12.91.164.116  |          8 |
| 12.91.164.106  |          7 |
| 12.91.165.211  |          7 |
| my.net.153.190 |          6 |
| my.net.110.52  |          6 |
| my.net.87.98   |          5 |
| my.net.146.55  |          4 |
| 66.32.232.141  |          3 |
| my.net.53.183  |          3 |
| my.net.53.179  |          2 |
| my.net.118.48  |          1 |
| my.net.153.145 |          1 |
| my.net.90.59   |          1 |
| my.net.182.9   |          1 |
| my.net.83.146  |          1 |
+----------------+------------+
```

Correlations:
From Bradley Urwiller's practical (http://www.giac.org/practical/Bradley_Urwiller_GCIA.pdf), he also indicates that snort logs by themselves do not provide enough information to gauge the veracity of these alerts, that only a true packet sniff would be able to determine whether these were true alerts or false positives.

Defensive Recommendations:

While it is possible that the top talker is someone on your internal network trying to run cgi exploits, the overall distribution tends to make me think these might be false positives. Snort can be configured to watch traffic over any port and you can use "--cginull" to disable detecting apparent null byte characters. It may be worth while disabling this and setting up a more specific rule that watches for null byte injections to your websites. It would also be worthwhile making certain all your internal hosts had updated versions of IE per the MS article mentioned above.

Also, setting up egress filters on a firewall to prevent unauthorized port 80 attempts, and auditing internal web site cgi's to ensure they aren't vulnerable to null byte poisoning, would be worthwhile.

4th Most Severe

Alert
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize

An IIS ISAPI Overflow ida attack makes reference to a vulnerability uncovered in IIS where it is possible to overflow the memory buffer in IIS by making special calls to .ida, and then inserting arbitrary code to be run on the system. Code Red makes special use of this vulnerability.

According to the CERT Advisory, Code Red conducts a number of tactics to penetrate systems. Infected hosts generate a random list of IP's and sweep those subnets for open port 80 hosts. Once they've found an available host, they then attempt to overflow an indexing buffer (See CA-2001-13 for details http://www.cert.org/advisories/CA-2001-13.html) by sending a specially crafted .ida request, triggering the vulnerability in Microsoft's IIS ISAPI.DLL. From there, it does a combination of denial of service, propagation, and hiding tactics.

The next question is, who are the big talkers?

```
mysql> select distinct(source_ip), count(*) as totalcount from alerts where
alert_name LIKE 'IDS552%' group by source_ip order by totalcount desc;
+-----------------+------------+
| source_ip       | totalcount |
+-----------------+------------+
| my.net.84.234   |     482601 |
| my.net.84.23    |       1109 |
```

I've purposefully truncated this log. It actually showed about a couple hundred other source IP's, but all from external networks and each with only 1 occurrence. The worry I have is that the alert name includes the tags "INTERNAL nosize". The original arachnids rule that looks for this (http://www.whitehats.com/info/IDS552) specifies datagram sizes over 239 bytes. See the rule as written for Arachnids below...

```
alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype:
system-or-info-attempt; reference: arachnids,552;)
```

This is to reduce the number of false positives and trigger only on what look like buffer overflows (typically represented by large size packets). If the IDS rules author changed it to not look at size, as the alert name indicates, then it could trigger on any .ida call, some of which could conceivably be legitimate. More on this in the defensive recommendation section.

So let's take a quick look at the primary alert generator, my.net.84.234. I want to know every alert they were the source of.

```
+-------------------------------------------------------+------------+
| alert_name                                            | totalcount |
+-------------------------------------------------------+------------+
| IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize |     482601 |
| Possible trojan server activity                       |          5 |
+-------------------------------------------------------+------------+
```

2 rows in set (18.87 sec)

my.net.84.83 shows something similar

```
+-------------------------------------------------------+------------+
| alert_name                                            | totalcount |
+-------------------------------------------------------+------------+
| IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize |       1109 |
| Possible trojan server activity                       |          1 |
+-------------------------------------------------------+------------+
```

The trojan server activity is definitely bad, but its not clear to me yet whether this is just alerting as to a scan or if this is actually an active trojan. Another SQL query not shown here reveals a number of external hosts doing what appears to be a tcp port 27374 scan, and the return traffic appears to be resets, in that I see a single inbound packet followed immediately by an outbound packet. So I don't think the trojan activity is related to the ISAPI alerts. More on this topic later.

I also did a search for distinct destination IP's, not shown here, and it indicated that the destination hosts were totally spread out, no definable pattern other than random, and there was no host that stood out as the top receiver from my.net.84.234. This is consistent with documented Code Red behavior.

I was also curious as to what distinct alerts were destined to my.net.84.234.

```
mysql> select distinct(alert_name), count(*) as totalcount from alerts
where dest_ip = 'my.net.84.234' group by alert_name order by totalcount
desc;
+----------------------------------------------+------------+
| alert_name                                   | totalcount |
+----------------------------------------------+------------+
| Watchlist 000220 IL-ISDNNET-990517           |         47 |
| Watchlist 000222 NET-NCFC                    |         44 |
| Possible trojan server activity              |          7 |
| IDS552/web-iis_IIS ISAPI Overflow ida nosize |          1 |
| spp_http_decode: IIS Unicode attack detected |          1 |
+----------------------------------------------+------------+
```

The Watchlist rules look like custom rules setup by the IDS administrator looking for activity from certain hosts. Unclear what that would indicate except that something is out of spec. It looks as if the IDS administrator had already detected improper traffic coming from this a watchlist network in a previous trace and was on the look out for it. Also note we have a very suspicious IIS Unicode attack. The IDS552 alert was caused by host 144.132.168.26 at 17:26:24 on Aug 4.

```
08/04-17:26:24.427121  [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize
[**] 144.132.168.26:3088 -> my.net.84.234:80
```

The first instance of my.net.84.234 sending out its own ISAPI overflow attempts is at 17:26:26 on Aug 4, just two seconds after being hit by 144.132.168.26. I would say this makes 144.132.168.26 as a fairly likely culprit of being the original infector of this host.

Correlations:
The cause and effect established above showing how the ISAPI alerts didn't start firing until directly after the my.net.84.234 host was targeted with an ISAPI buffer overflow itself is a fairly good correlation. The random nature of the host selection by the infected my.net.84.234 correlates to the CERT Advisories noted above.

Defensive Recommendations:

Check both boxes to ensure proper ISAPI patches have been installed according to the associated Microsoft article MS01-044 (http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-044.asp). Determine if certain instances of IIS can be disabled altogether. Determine if its possible blocking port 80 on a packet filtering firewall. Scrub the two hosts for any sign of compromise, whether Code Red or anything else (possibly Nimda). In this case, internal infection is not likely since, if any hosts had been subsequently infected with Code Red inside the network, we should have seen them attempting to get outside via port 80 in the same manner as we saw my.net.84.234 and my.net.84.23.

Also, egress filters on a firewall would prevent outbound connect attempts from unauthorized hosts, thus preventing the spread of Nimda to external networks.

Details of Code Red and patching information can be found at the CERT Coordination Center, bulletin CA-2001-19, located at http://www.cert.org/advisories/CA-2001-19.html.

5th Most Severe

Alert
IRC evil - running XDCC

IRC, or Internet Relay Chat, was a precursor to instant messaging and an obvious follow through from bbs style chat rooms. XDCC adds file upload/download functionality to IRC enabling members of a chat room to exchange files with each other. Presumably, this rule attempts to catch those who are swapping files with each other.

A list of source IP's that triggered this rule

```
mysql> select distinct(source_ip), count(*) as totalcount from alerts where
alert_name LIKE 'IRC evil%' group by source_ip order by totalcount desc;
+----------------+------------+
| source_ip      | totalcount |
+----------------+------------+
| my.net.82.130  |       1170 |
| my.net.178.199 |        372 |
| my.net.82.87   |        160 |
| my.net.163.93  |        131 |
| my.net.178.172 |        101 |
| my.net.80.149  |         81 |
| my.net.163.78  |         37 |
| my.net.80.14   |          2 |
| my.net.162.226 |          1 |
+----------------+------------+
```

Some sample alerts taken from the raw logs...

```
08/01-02:27:27.374427  [**] IRC evil - running XDCC [**] my.net.163.93:4169 -
> 66.250.105.173:6667
```

```
08/01-02:31:50.552258  [**] IRC evil - running XDCC [**]
my.net.178.199:4150 -> 206.246.167.130:6667
08/01-02:38:36.866603  [**] IRC evil - running XDCC [**] my.net.80.149:3543 -
> 216.232.40.20:6667
08/01-02:54:37.034011  [**] IRC evil - running XDCC [**] my.net.80.149:3543 -
> 216.232.40.20:6667
08/01-03:03:29.920513  [**] IRC evil - running XDCC [**] my.net.163.78:4642 -
> 203.116.84.127:6667
08/01-03:08:40.297382  [**] IRC evil - running XDCC [**] my.net.163.93:3438 -
> 66.250.105.173:6667
```

I notice that the source ports are all ephemeral, so this is probably *not* a false positive triggering because of random ephemeral port selection for 6667.

I decided to do an nslookup on one of these destinations. For 66.250.105.173...

Name: hm2k.org
Address: 66.250.105.173

I went to www.hm2k.org just to see what kind of site it was. This was the intro banner.

Welcome HM2K.ORG! Here you will find Developments - most of them are IRC related, but I have more to come soon. Music - music has a big influence of my life, when i'm not on my computer, i'm spinning on the decks. Tools - There are a couple of useful tools here, nothings special, more to come soon! Contact - there are a few ways you can contact me

This very much looks like authentic IRC style conversations. It is possible file trading is happening on some/all of these hosts assuming the snort rule is set up properly (which from all appearances seems that it is set up properly).

Defensive Recommendations:
Confirm what network policy is pertaining to XDCC IRC style chat and deploy a firewall preventing that kind of communication from occurring. If this kind of traffic is permitted, adjust snort rule to reflect the policy so it won't show in the logs.

6th Most Severe

Alert
Possible trojan server activity

Even though I hinted in the ISAPI discussion that the trojan server activity may just be an indicator that its a scan and not actually trojan's active, it warrants a look just to be sure.

I first wanted to see what port they were triggering on.

```
mysql> select distinct(source_port), count(*) as totalcount from alerts
where alert_name = 'Possible trojan server activity' group by source_port
order by totalcount desc;
+-------------+------------+
| source_port | totalcount |
+-------------+------------+
| 27374       |       1809 |
| 143         |         12 |
| 3137        |          9 |
| 4938        |          8 |
| 1293        |          8 |
| 2357        |          7 |
| 4050        |          7 |
```

I truncated the logs because they went on for a long time and had no definable pattern after the first entry of 27374. The ports after 27374 looked like ephemeral (i.e. above 1023) ports that TCP will choose when establishing a TCP session. This tells me two things. One, the attacker(s) were looking for open 27374 ports, which is the typical port SubSeven will listen on. Two, the snort rule was logging every instance of port 27374, not caring what flags were set or whether it was a source or destination port.

Next I wanted to find out what the destination IP distribution looked like and look specifically for home network IP's.

```
mysql> select distinct(dest_ip), count(*) as totalcount from alerts where
alert_name = 'Possible trojan server activity' group by dest_ip order by
totalcount desc;
+-----------------+------------+
| dest_ip         | totalcount |
+-----------------+------------+
| 63.196.247.234  |        484 |
| 80.62.155.240   |        298 |
| 61.102.149.115  |        291 |
| 217.136.63.141  |        291 |
| 218.154.202.148 |        203 |
| 138.88.40.155   |         86 |
| 66.76.134.169   |         74 |
| 66.21.144.203   |         34 |
```

Doing spot checks of destination addresses for these hosts, it looks like this was just a scan. Internal hosts got hit from 1 to 6 times, responding only when queried by an attacker. At this point, I'm not inclined to feel this was actual trojan activity, very likely just the result of a scan because there is no sign of prolonged conversation and because the attacking hosts seemed intent on hitting as many hosts in the network as possible. The one bad thing about this is there were replies to these requests, and its not clear whether they were SYN-ACK's or just RST's, but either way this would indicate there was no firewalling off port 27374.

Correlations:

I checked the scan logs. It seemed reasonable since I was supposing this was a scan.

I grepped for all instances of 27374, it returned logs for many but not all of the above mentioned hosts. It still looks like scan activity.

Defensive Recommendations:

Adjust snort rules to look for 27374 SYN-ACK packets from internal network outbound, indicating possible establishment of TCP 3-way handshake. Simply triggering on any packet will generate inordinate number of false positives. Also deploy a firewall to filter out this traffic.

7th Most Severe

Alert
beetle.ucs

No clue about this alert, but it is my suspicion it is a network specific alert, something custom written by the admin in the hopes of catching something.

By looking at each alert entry separately, it seems the one thing they each have in common is the traffic is either to or from my.net.70.69. An Arin WhoIs Lookup shows the following about the host...

```
OrgName:    University of <OMITTED BY IDS ANALYST>
OrgID:      U<OMITTED>

NetRange:   my.net.0.0 - my.net.255.255
CIDR:       my.net.0.0/16
```

An NSLOOKUP on the IP in question returns...

```
Name:    beetle.ucs.u<OMITTED>.edu
Address:  my.net.70.69
```

The host name is probably how the rule name was derived. There doesn't seem to be any pattern to the ports the alert is tripping on. Without more information, it would be impossible for me to determine whether this was hostile or not.

Defensive Recommendations:

See if this rule can't be narrowed to flag suspicious behavior only. As is, it has no meaning to anyone outside of whoever set the rule up.

8th Most Severe

Alert(s)

Port 55850 tcp - Possible myserver activity - ref. 010313-1
Port 55850 udp - Possible myserver activity - ref. 010313-1

Myserver is an older DDOS tool that comes with its own set of root kit tools designed to hide its presence from a server administrator. It commonly talks over port 55850. A true positive of this alert would indicate possible master/slave communication indicating someone were passing instructions to the infected machine.

Looking at source and destination ports of these logs, I can see that the 55850 is paired up with commonly used Internet ports, like port 80, 21, and 25. Since I have no information about the TCP flag for any of these alerts, I'm inclined to assume that these were false positives generated because, after approximately 60,000 connections, eventually a host will pick 55850 as its ephemeral source port.

The rule for UDP is a little bit tougher since it doesn't use state information in determining the condition of a session. Looking at these alert logs for UDP, we see the following...

```
ALERT  Aug   01   05:02:15.986851   128.8.7.3        55850        my.net.140.9    33456
ALERT  Aug   01   05:02:15.990575   128.8.7.3        55850        my.net.140.9    33457
ALERT  Aug   01   05:02:15.992542   128.8.7.3        55850        my.net.140.9    33458
ALERT  Aug   01   06:03:16.547178   62.2.172.99      55850        my.net.70.207   12300
ALERT  Aug   01   06:03:16.565780   my.net.70.207    12300        62.2.172.99     55850
ALERT  Aug   02   03:39:52.183640   130.215.5.33     55850        my.net.140.9    33462
ALERT  Aug   02   03:39:52.197677   130.215.5.33     55850        my.net.140.9    33463
ALERT  Aug   02   05:11:32.948633   155.101.16.5     55850        my.net.140.9    33483
ALERT  Aug   02   05:11:33.059132   155.101.16.5     55850        my.net.140.9    33485
ALERT  Aug   02   13:38:00.162275   63.250.205.26    41281        my.net.114.110  55850
ALERT  Aug   02   17:23:10.666109   129.59.1.201     55850        my.net.140.9    33465
ALERT  Aug   02   17:23:10.711616   129.59.1.201     55850        my.net.140.9    33467
ALERT  Aug   03   07:53:26.666436   128.109.100.100  55850        my.net.140.9    33459
ALERT  Aug   03   07:53:26.692058   128.109.100.100  55850        my.net.140.9    33460
ALERT  Aug   03   07:53:26.718058   128.109.100.100  55850        my.net.140.9    33461
ALERT  Aug   03   09:15:35.357279   204.183.84.240   55850        my.net.137.7    88
ALERT  Aug   03   09:15:35.362793   my.net.137.7     88           204.183.84.240  55850
ALERT  Aug   05   07:05:32.383401   63.241.203.98    13398        my.net.117.25   55850
```

Note that the only time one of our internal hosts communicates to an external host over 55850 is as a response, never as an initiator. Also, many of the ports in the 33000 range would indicate a possible UNIX style traceroute (Microsoft traceroutes use ICMP to determine the hop path, *nix style hosts use UDP packets using destination ports in the 33000 range).

<u>Defensive Recommendations:</u>

These are probably false positives. Narrow this rule to specify destination port 55850 with the SYN flag set. That should remove the chance of picking up source port 55850. Also, deploy a packet filtering firewall to block inbound or outbound connection attempts to port 55850 either TCP or UDP. Consider blocking ICMP traceroutes inbound as well as any unauthorized UDP traffic.

9th Most Severe

Alert
High port 65535 tcp - possible Red Worm - traffic
High port 65535 udp - possible Red Worm - traffic

Red Worm, also known as Adore Worm, attempted to gain access to systems running vulnerable versions of BIND, LPRng, and rpc-statd. Once it is able to infect a machine, it then creates a backdoor listening on TCP port 65535. Details about Adore/Red Worm can be found at http://www.digital-minds.org/Network/adore_worm.pdf.

So, the question now is, the traffic that generated this alert, does it look like backdoor type traffic? A SQL query to determine source IP's and counts...

```
mysql> select distinct(source_ip), count(*) as totalcount from alerts where
alert_name LIKE 'High port 65535%' group by source_ip order by totalcount
desc;
+-----------------+------------+
| source_ip       | totalcount |
+-----------------+------------+
| my.net.83.150   |        107 |
| my.net.165.24   |         76 |
| 66.79.48.188    |         34 |
| 217.226.116.163 |         29 |
| 80.14.16.127    |         28 |
| my.net.83.146   |         27 |
| 63.241.203.98   |         24 |
| 12.129.73.230   |         23 |
| 62.143.16.85    |         19 |
```

So, my.net.83.150 and my.net.165.24 seem particularly active with this. A quick look at the alert logs for my.net.83.150...

```
08/04-21:03:33.940020  [**] spp_portscan: portscan status from
my.net.83.150: 2 connections across 2 hosts: TCP(0), UDP(2) [**]
08/04-21:03:35.564352  [**] spp_portscan: portscan status from
my.net.83.150: 2 connections across 2 hosts: TCP(0), UDP(2) [**]
08/04-21:03:37.436742  [**] spp_portscan: End of portscan from
my.net.83.150: TOTAL time(40s) hosts(39) TCP(0) UDP(58) [**]
08/04-20:49:01.099627  [**] High port 65535 udp - possible Red Worm -
traffic [**] 80.14.16.127:65535 -> my.net.83.150:6257
08/04-20:49:01.740064  [**] High port 65535 udp - possible Red Worm -
traffic [**] my.net.83.150:6257 -> 80.14.16.127:65535
08/04-20:49:02.872213  [**] High port 65535 udp - possible Red Worm -
traffic [**] 80.14.16.127:65535 -> my.net.83.150:6257
08/04-20:49:02.873192  [**] High port 65535 udp - possible Red Worm -
traffic [**] my.net.83.150:6257 -> 80.14.16.127:65535
08/04-20:49:04.033266  [**] High port 65535 udp - possible Red Worm -
traffic [**] 80.14.16.127:65535 -> my.net.83.150:6257
```

Couple of points of interest. First, our snort scan preprocessor thinks that this host is scanning external hosts for udp 65535. Second, the source port in all cases was 6257. TCP and UDP port 6257 correlates to a program called WinMX, a peer to peer file sharing program (see http://www.winmx.com/ for details). However, there is no direct link between 65535 and 6257. Host my.net.83.150 does not seem to increment its source ports during its various connect attempts, so its very likely a program or tool making this traffic (possibly WinMX).

Also, no other alerts were generated from this host other than previously discussed possible trojan activity (which is in all likelihood related to the above mentioned scans). No BIND, LPRng, or rpc.statd alerts were generated by attacks to this host either.

Host my.net.165.24 shows similar behavior

Correlations:

From Jeff Zahr in his GCIA practical, (http://www.giac.org/practical/Jeff_Zahr_GCIA.doc), he noticed similar traffic and also referenced WinMX in his analysis as a likely culprit.

Defensive Recommendations

Determine if such traffic is against network policy and deploy a firewall to filter out unwanted traffic.

Scans Methodology

Sifting through the scan logs of a network are less important than the alert logs, but they still can show some very useful information, not only to correlate the alert logs, but also to pick up on things the alert logs might have missed. Scan logs can provide an analyst with the feel of a network, how much activity is it seeing, what kinds of threats is it showing. To view the scans and to help me decide what to examine, I ran an SQL query looking for distinct destination ports and their total counts. What follows is a truncated list.

```
mysql> select distinct(dest_port), count(*) as totalcount from scans group
by dest_port order by totalcount desc limit 100;
+-----------+------------+
| dest_port | totalcount |
+-----------+------------+
| 41170     |    2442717 |
| 80        |     815894 |
| 6257      |     204314 |
| 1433      |      72379 |
| 21        |      35331 |
| 28800     |      29492 |
| 53          |       17388 |
| 27005       |       16382 |
| 139         |       16185 |
| 7003        |       14915 |
| 6970        |       13408 |
| 445         |       12692 |
| 111         |       12320 |
```

Port 80 can probably be accounted for by all the Nimda and Code Red activity shown in the above analysis. Port 6257 correlates to the WinMX discussions. But the 41170 deserves some focus as I'm not immediately familiar with this and do not recall seeing this port show up in any previous searches. More on this a little further on.

1433 and 21 are SQL and FTP, fairly standard scans. 28800 correlates to a Microsoft Network Gaming port, very likely the result of someone playing such games from within the network. Port 53 is DNS, it is possible that the IDS Analyst would want to check this to see if snort is ignoring his/her internal DNS servers in the port scan detector, thus reducing the likelihood of false positives here.

Port 27005 correlates to an online game called Counter-Strike, again, likely internal hosts playing online games. Port 139 and 445, netbios, again, fairly commonly scanned port. 7003 I was unable to find in my research. 6970 correlates to the trojan GateCrasher. 111 is the infamous RPC, again, very common.

Scan - Port 41170

The top scan for all five days was for destination port 41170, easily triple the traffic generated by all of the Nimda and Code Red port 80 scans. Port 41170 correlates to Blubster, a peer-to-peer file sharing program found at www.blubster.net. These scans would therefore indicate possible peer to peer file sharing occurring within the home network.

The following source IP's were shown to have some scan activity with 41170.

```
mysql> select distinct(source_ip), count(*) as totalcount from scans where
dest_port = '41170' group by source_ip order by totalcount desc limit 100;
+---------------+------------+
| source_ip     | totalcount |
+---------------+------------+
| my.net.70.200 |    2436774 |
| my.net.70.180 |       5916 |
| 63.241.203.98 |         13 |
| 193.252.2.179 |          8 |
| 64.38.251.140 |          5 |
| my.net.137.7  |          1 |
+---------------+------------+
```

Searching for unique destination IP's showed a tremondous number of hosts, each with a fairly even spread as far as scan counts are concerned (much what you'd expect from a peer-to-peer file sharing system). Either way, it looks like host my.net.70.200

and my.net.70.180 are worth an extra look.

Scan Logs Defensive Recommendations:

Just by looking at the scan overview, it is easy to see that much is happening in the way of vulnerability and trojan reconnaissance, as well as online gaming, not at all surprising at a University. Peer to peer file sharing, gaming, and other such recreational activities can have a very severe impact on a network's performance, which degrades the usefulness of the network as a tool for research and learning. If network performance is a concern, consider presenting an argument to the powers that be to selectively turn off or throttle with QOS or some other kind of packet prioritizer the major talking ports to free up bandwidth.

Out of Spec Methodology

These logs indicate improper packet construction, indicating improper or unexpected header sequence. An alert here does not immediately indicate an attack, just like in any other log.

Aug 1, 4, and 5 showed very low instances of OOS alerts. Aug. 2 and 3, however, showed considerable alerts. However, all of the logs had a single unusual element in common, they all shared a TCP Flag set of 21S*****. An example of the alert is as follows.

OOS - 68.32.126.64

The OOS file for Aug 2 had a much higher count than any of the other OOS files for the 5 day span. Of the 1124 alerts generated in this file, 396 pertained to 68.32.126.64. The following is a sample of what was seen.

```
08/01-11:12:24.004671 68.32.126.64:27153 -> MY.NET.6.7:110 TCP TTL:48 TOS:0x0 ID:65515  DF
21S***** Seq: 0xE3302C37   Ack: 0x0   Win: 0x16D0 TCP Options => MSS: 1460 SackOK TS:
55890908 0 EOL EOL EOL EOL

08/01-11:14:31.228286 68.32.126.64:27155 -> MY.NET.6.7:110 TCP TTL:48 TOS:0x0 ID:41952  DF
21S***** Seq: 0xEAE0A66E   Ack: 0x0   Win: 0x16D0 TCP Options => MSS: 1460 SackOK TS:
55903630 0 EOL EOL EOL EOL

08/01-11:27:15.546629 68.32.126.64:27170 -> MY.NET.6.7:110 TCP TTL:48 TOS:0x0 ID:29061  DF
21S***** Seq: 0x1A5D4199   Ack: 0x0   Win: 0x16D0 TCP Options => MSS: 1460 SackOK TS:
55980060 0 EOL EOL EOL EOL

08/01-11:28:19.000174 68.32.126.64:27171 -> MY.NET.6.7:110 TCP TTL:48 TOS:0x0 ID:59936  DF
21S***** Seq: 0x1EA6A2C5   Ack: 0x0   Win: 0x16D0 TCP Options => MSS: 1460 SackOK TS:
55986405 0 EOL EOL EOL EOL
```

Destination Port 110 correlates to Pop3 services. However, the odd thing about these packets is that the ECN bits in the "reserved" section of the tcp header were set. This is usually not set, since ECN is a relatively new standard that hasn't been widely adopted.

The RFC that governs ECN, 2481 (http://www.ietf.org/rfc/rfc2481.txt?number=2481), indicates in section 5 the following...

> We propose that the Internet provide a congestion indication for incipient congestion (as in RED and earlier work [RJ90]) where the notification can sometimes be through marking packets rather than dropping them. This would require an ECN field in the IP header with two bits. The ECN-Capable Transport (ECT) bit would be set by the data sender to indicate that the end-points of the transport protocol are ECN-capable. The CE bit would be set by the router to indicate congestion to the end nodes. Routers that have a packet arriving at a full queue would drop the packet, just as they do now.

...Thus, I would expect the TOS of these packets to be something other than 0 to properly indicate ECN. However, the TOS on all packets is 0. This would indicate a true out of spec packet that could be used by a would be attacker, possibly to evade an IDS that was set to trigger an alert on a very specific flag setting.

No alerts were generated from or to this address.

Correlations

From Judy Novak (http://archives.neohapsis.com/archives/snort/2000-09/0078.html) in a mailing list post, she indicates that ECN traffic should have a TOS set to a non-zero value, and that any such traffic that shows ECN where the TOS is zero should be regarded as suspicious.

Defensive Recommendations

For this one particular alert, determine if POP3 is an allowed protocol into the network. Then, determine if your firewall can be configured to drop out of spec ECN packets before they enter the network. Since all of the alerts in the OOS logs showed a similar TCP flag set but going to different ports, it would be worthwhile investigating if there is some way to drop these packets before they enter the network.

Overall Recommendations

Based on the above findings, there are a number of items that stand out as common elements that could be applied to mitigate exposure.

1. Packet Filtering Firewall - Many problems can be avoided simply by taking the time

to deploy a firewall that blocks improper inbound and outbound connections. Many of the alerts and scans indicated the lack of any proper firewalling. A decent perimeter defense is a proper first step to securing the network.

If it is not practical for the University to deploy a network-wide firewall, it would very much be worth the time and effort to identify core University servers and mainframes and segment those off behind a firewall. This would help mitigate the risk of compromise to machines that could potentially house sensitive data, and it would allow students and other users unfettered access to the Internet.

2. Server patches - The time from when a vulnerability is announced to someone creating an automated exploit is getting shorter and shorter. This means staying as up to date as possible on patches and fixes for problems found. It also means having a defined methodology for deploying new servers, like a security checklist. Microsoft provides many examples, such as this one for Windows 2000 Server at http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/chklist/w2ksvrcl.asp, of generic checklists that, had they been employed on this network, could have prevented many of the problems with compromised hosts. Once you've identified your core servers, stay up to date with fixes from software vendors and have a strategy on the board that addresses who, how, and when security updates would be applied.

3. Fine tune SNORT Rules - Snort can generate a tremendous wealth of information, but it can generate even more useless clutter. Take time to fine tune the rules of SNORT to look only for activity that is 100% bad (or as close to it as you can get). Specify flags, datagram sizes, content strings, or anything that makes an exploit different than expected traffic. This should greatly improve the efficiency of the IDS analyst going through the logs and making recommendations.

Registration Information on Five Selected IP Addresses

1. 194.98.189.139

Selected because if its involvement with severe alert item 3 and RPC/STATDX attempts. Very likely this host was attempting to compromise one our the internal hosts using the STATDX buffer overflow.

From www.ripe.net

```
inetnum:      194.98.189.128 - 194.98.189.143
netname:      INGENCYS-NET1
descr:        INGENCYS
```

```
country:     FR
admin-c:     DR5-RIPE
tech-c:      JB371-RIPE
status:      ASSIGNED PA
remarks:     abuse@fr.uu.net
mnt-by:      IWAY-NOC
changed:     frederic.martzel@mciworldcom.fr 20010924
source:      RIPE

route:       194.98.0.0/16
descr:       UUNET-BLOCK1
descr:       UUNET France Block 1
origin:      AS702
remarks:     *************************************
remarks:     For all spamming or hacking problems
remarks:     please send your requests directly to
remarks:     abuse@fr.uu.net
remarks:     *************************************
notify:      net-adm@mciworldcom.fr
mnt-by:      IWAY-NOC
changed:     net-adm@iway.fr 19981109
changed:     frederic.martzel@mciworldcom.fr 20011114
source:      RIPE

role:        technical contact
address:     UUNET FRANCE
address:     215, Avenue Georges Clemenceau
address:     F-92024 NANTERRE Cedex
phone:       +33 1 56 38 22 00
fax-no:      +33 1 56 38 22 01
e-mail:      net-adm@mciworldcom.fr
admin-c:     VP1616-RIPE
admin-c:     FM7174-RIPE
admin-c:     AW7486-RIPE
tech-c:      ZM321-RIPE
tech-c:      AH6610-RIPE
tech-c:      TC334-RIPE
nic-hdl:     JB371-RIPE
remarks:     ------------------------------------
remarks:     For all spamming or hacking problems
remarks:     please send your requests directly to
remarks:     abuse@fr.uu.net
remarks:     ------------------------------------
mnt-by:      IWAY-NOC
changed:     frederic.martzel@mciworldcom.fr 20010828
source:      RIPE
```

```
person:      Monsieur De Royer
address:     INGENCYS
address:     4, Rue de la Madeleine
address:     45140 ST JEAN DE LA RUELLE, France
phone:       +33 2 37 25 12 00
fax-no:      +33 2 37 25 12 00
nic-hdl:     DR5-RIPE
mnt-by:      IWAY-NOC
changed:     frederic.martzel@mciworldcom.fr 20010924
source:      RIPE
```

2. 61.182.50.241

Selected for the same reasons as the first. From www.apnic.net

```
inetnum:     61.182.0.0 - 61.182.255.255
netname:     CHINANET-HE
descr:       CHINANET Hebei province network
descr:       Data Communication Division
descr:       China Telecom
country:     CN
admin-c:     DK26-AP
tech-c:      ZC24-AP
mnt-by:      MAINT-CHINANET
mnt-lower:   MAINT-CHINANET-HE
changed:     hostmaster@ns.chinanet.cn.net 20010216
status:      ALLOCATED PORTABLE
source:      APNIC

person:      Dongmei Kou
address:     A12,Xin-Jie-Kou-Wai Street,
address:     Beijing,100088
country:     CN
phone:       +86-10-62370437
fax-no:      +86-10-62053995
e-mail:      chunguangcanlanxiaobajie@sina.com
nic-hdl:     DK26-AP
mnt-by:      MAINT-NEW
changed:     chunguangcanlanxiaobajie@sina.com 20010405
source:      APNIC

person:      zhiyong chen
address:     hebei province shijiazhuang
address:     fanxi road No.19
```

```
address:      hebei shuju tongxin ju
country:      CN
phone:        +86-311-6051394
fax-no:       +86-311-6672895
e-mail:       jixin@sj-user.he.cninfo.net
nic-hdl:      ZC24-AP
mnt-by:       MAINT-CHINANET-HE
changed:      chenzhy@public.sj.he.cn 20010212
source:       APNIC
```

3. 61.145.69.74

This looks to be the likely source of our Nimda compromise detailed in 2nd Most Severe Alert section. The following information was obtained from www.apnic.net

```
inetnum:      61.145.0.0 - 61.145.255.255
netname:      CHINANET-GD
descr:        CHINANET Guangdong province network
descr:        Data Communication Division
descr:        China Telecom
country:      CN
admin-c:      CH93-AP
tech-c:       WM12-AP
mnt-by:       MAINT-CHINANET
mnt-lower:    MAINT-CHINANET-GD
changed:      hostmaster@ns.chinanet.cn.net 20000701
status:       ALLOCATED PORTABLE
source:       APNIC

person:       Chinanet Hostmaster
address:      No.31 ,jingrong street,beijing
address:      100032
country:      CN
phone:        +86-10-66027112
fax-no:       +86-10-66027334
e-mail:       hostmaster@ns.chinanet.cn.net
nic-hdl:      CH93-AP
mnt-by:       MAINT-CHINANET
changed:      hostmaster@ns.chinanet.cn.net 20020814
source:       APNIC

person:       WU MIAN
address:      NO.1,RO.DONGYUANHENG,YUEXIUNAN,GUANGZHOU
country:      CN
phone:        +086-20-83877223
```

```
fax-no:      +86-20-83877223
e-mail:      ipadm@gddc.com.cn
nic-hdl:     WM12-AP
mnt-by:       MAINT-CHINANET-GD
changed:       ipadm@gddc.com.cn 20010820
source:       APNIC
```

4. 211.141.120.18

Chosen for the similar reasons as 61.145.69.74, is a likely source of the Nimda attack. Again, obtained from www.apnic.net

```
inetnum:      211.141.120.16 - 211.141.120.31
netname:       JAWMNET
descr:       WUMING Network bar =20
descr:       provide Internet accessing service
descr:       JI AN city, Jiangxi Province=20
country:      CN
admin-c:      XW45-AP
tech-c:      SS170-AP
mnt-by:       MAINT-CN-CMCC
changed:       hostmaster@chinamobile.com 20011128
status:                ASSIGNED NON-PORTABLE
source:       APNIC
changed:       hm-changed@apnic.net  20020827

person:       Xiaoyun Wang
address:      53A,Xibianmennei Ave.,Xuanwu District,Beijing,100053 China
country:      CN
phone:        +86-10-6360-0159
fax-no:      +86-10-6360-0117
e-mail:      wangxiaoyun@chinamobile.com
nic-hdl:     XW45-AP
mnt-by:       MAINT-CN-CMCC
changed:       sunshaoling@chinamobile.com 20010831
source:       APNIC

person:       Shaoling Sun
address:      53A,Xibianmennei Ave.,Xuanwu District,Beijing,100053 China
country:      CN
phone:        +86-10-6360-0159
fax-no:      +86-10-6360-0117
e-mail:      sunshaoling@chinamobile.com
nic-hdl:     SS170-AP
mnt-by:       MAINT-CN-CMCC
```

```
changed:     sunshaoling@chinamobile.com 20010831
source:      APNIC
```

5. 216.228.171.81

I selected this one because it was the top source IP address for scans generated that were not from internal hosts. It seemed to be very interested about Microsoft SMB style traffic, ports 137, 139, and 445. I obtained the following information from www.arin.net

```
Search results for: 216.228.171.81

Bend Cable BENDCABLE (NET-216-228-160-0-1)
                              216.228.160.0 - 216.228.191.255
bend cable communications BCCI228-DOCSIS (NET-216-228-168-0-1)
                              216.228.168.0 - 216.228.172.255
```

List of References

Jacobowitz, Daniel. "Multiple Linux Vendor rpc.statd Remote Format String Vulnerability." 16 Jul. 2000. URL: http://online.securityfocus.com/bid/1480/discussion (2 Oct. 2002).

"Microsoft Security Bulletin MS01-058." 14 Feb. 2002. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-058.asp (2 Oct. 2002).

Danyliw, Roman and Allen Householder. " CERT® Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL." 17 Jan. 2002. URL: http://www.cert.org/advisories/CA-2001-19.html (2 Oct. 2002).

Ellis, Joe. "GCIA Practical Assignment, v3.0." 14 May 2002. URL: http://www.giac.org/practical/Joe_Ellis_GCIA.doc (2 Oct 2002).

??? [4] http://www.giac.org/practical/Joe_Ellis_GCIA.doc

Beadsley, Tod. "Intrusion Detection and Analysis: Theory, Techniques, and Tools." 8 May 2002. URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc (2 Oct. 2002).

Urwiller, Bradley D. "Practical Assignment Version 3.0." 23 Apr. 2002. URL:

www.giac.org/practical/Bradley_Urwiller_GCIA.pdf (2 Oct. 2002).

" CERT® Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service DLL." 17 Jan. 2002. URL: http://www.cert.org/advisories/CA-2001-13.html (2 Oct. 2002).

"Windows 2000 Server Baseline Security Checklist." URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/chklist/w2ksvrcl.asp (2 Oct. 2002).

Rain Forest Puppy. "Perl CGI Problems." 27 Feb. 2001. URL: http://www.wiretrip.net/rfp/p/doc.asp/i2/d6.htm (2 Oct. 2002).

Plummer, David C. "An Ethernet Address Resolution Protocol." Nov 1982. URL: http://www.ietf.org/rfc/rfc0826.txt?number=826 (3 Oct. 2002).

"Ettercap" http://ettercap.sourceforge.net/ (3 Oct. 2002).

Tripunitara, Mahesh, and Partha Dutta. "A Middleware Approach to Asynchronous and Backward Compatible Detection and Prevention of ARP Cache Poisoning." URL: http://www.acsac.org/1999/papers/fri-b-0830-dutta.pdf (3 Oct. 2002).

URL: ftp://ftp.ee.lbl.gov/arpwatch.tar.Z (3 Oct. 2002).

Rain Forest Puppy, "A look at whisker's anti-IDS tactics." 24 Dec. 1999. URL: http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html (3 Oct. 2002).

Fielding, R. et. al. "Hypertext Transfer Protocol -- HTTP/1.1." Jun. 1999 URL: http://www.ietf.org/rfc/rfc2616.txt?number=2616 (3 Oct. 2002).

Tzu, Sun "The Art of War" URL: http://www.art-of-war.com (3 Oct. 2002).

"Frontpage Publishing DoS (Denial of Service)" 22 Dec. 2000. URL: http://lists.insecure.org/win2ksecadvice/2001/Jan/0010.html (4 Oct. 2002).

Stearns, William. "Passive OS Fingerprinting Tool." 2001 URL: http://www.stearns.org/p0f/devel/p0f.fp (3 Oct. 2002).

Frederick, Karen Kent. "Network Intrusion Detection Signatures, Part Two." 22 Jan. 2002. URL:
http://online.securityfocus.com/infocus/1534 (4 Oct. 2002).

Yuen, Rick Winkey. "GIAC Intrusion Detection, Practical Assignment." 11 Oct. 2001 URL: http://www.giac.org/practical/Rick_Yuen_GCIA.doc (4 Oct. 2002).

Aronne, Eugene J. "The Nimda Worm: An Overview." 8 Oct. 2001. URL: http://rr.sans.org/malicious/nimda3.php (4 Oct. 2002).

" Microsoft Security Bulletin MS01-044." 15 Aug. 2001. URL: http://www.microsoft.com/technet/security/bulletin/MS01-044.asp (4 Oct. 2002).

Dr. SuSE, " IDS552 'IIS ISAPI OVERFLOW IDA.'" URL: http://www.whitehats.com/info/IDS552 (4 Oct. 2002).

URL: http://www.hm2k.org (4 Oct. 2002).

Ramakrishnan, K. "A Proposal to add Explicit Congestion Notification (ECN) to IP." Jan. 1999. URL: http://www.ietf.org/rfc/rfc2481.txt?number=2481 (4 Oct. 2002).

Novak, Judy. "[Snort-users] RE: Castor's use of 'ECN' shut-off." 15 Sep. 2000. URL: http://archives.neohapsis.com/archives/snort/2000-09/0078.html (4 Oct. 2002).

Dell, J. Anthony. " Adore Worm – Another Mutation." 6 Apr. 2001. URL: http://www.digital-minds.org/Network/adore_worm.pdf (4 Oct. 2002).

URL: http://www.winmx.com (4 Oct. 2002).

Zahr, Jeff. "SANS GIAC Intrusion Detection In Depth." 15 Nov. 2001. URL: http://www.giac.org/practical/Jeff_Zahr_GCIA.doc (4 Oct. 2002).