Global Information Assurance Certification Paper

Intrusion Detection
In-Depth

By John M. Melvin

Table of Contents

Assignment # 1: Describe the State of Intrusion Detection

ICMP Traffic: Cat vs. Mouse

(No-Mice) Normal, Obliging, Malicious, Intrusive, Covert, and Evasive

I guess we can describe ICMP in many ways, and since it's integrated with IP it's bound by those rules and behaviors, and plenty of us have found out how to take advantage of they way protocols should behave. Remember when ICMP was ignored and thought of as just administrative messages or testing? Only recently have we begun to block incoming pings and other ICMP traffic, but it's still a very popular way to carry out attacks, and it's still a very useful tool when troubleshooting a network. So, we have a cat and mouse situation; we like ICMP and most of us need it to conduct network analysis, so we can't block it entirely. At the same time, by allowing it we become vulnerable to many possible ways it can be used against us. Just earlier this year (July) the Black Hat convention held a class on this subject, so let's not put this on the back burner again and assume we have everything covered with perimeter devices and firewalls. I decided to cover this topic because of the many ways ICMP can be used, the fact it's still widely misunderstood, and because I wanted to take some time and learn about it so I can become more confident when analyzing it. The scope of this assignment is to cover reconnaissance techniques, and a concentration on how we can use intrusion detection to discover operating systems using ICMP messages (this is not a paper on the type of exploits and tools used, for instance a Smurf attack, TFN, Loki, or firewalk, unless those tools are needed to create scans and traces). If you want to research and learn about various tools that can implement ICMP attacks take a look at some good papers I've listed below:
http://rr.sans.org/threats/ICMP_attacks.php
http://www.sans.org/newlook/resources/IDFAQ/smurf.doc
http://www.whitehats.ca/main/members/Jeff/gcia_assign_2/gcia_assign_2.html

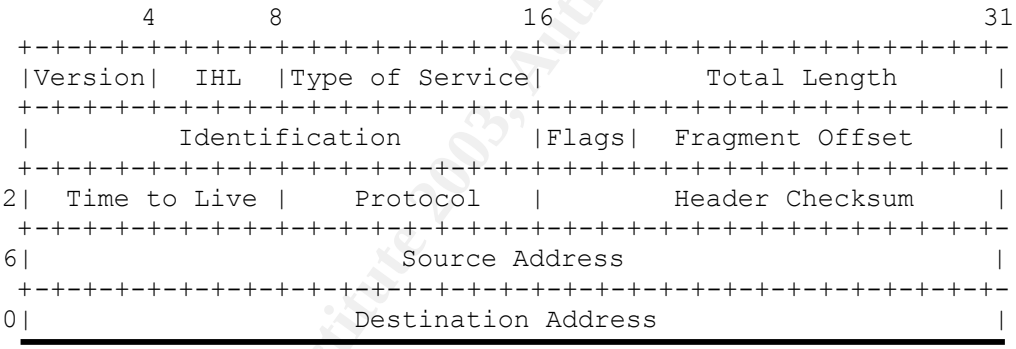
Let's now take a look at how ICMP is used, how it should be used, and how it has become a powerful attack tool.

The ICMP Protocol

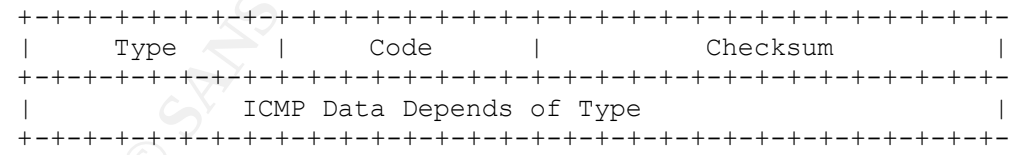
The ICMP Protocol follows RFC 792 and referenced in other RFC's specific to their function, for example:
RFC 1191 - Path MTU Discovery
RFC 1256 - Router Discovery
RFC 896 - Source Quench
RFC 950 - Address Mask Extensions
RFC 1122 - Communication Layers)
RFC 1812 - IP version 4 Routers
RFC 1349 - Type of Service in the Internet Protocol Suite1

ICMP gives us a way to receive error messages to troubleshoot a network (like when a host or router tells the source about errors in delivery) and tools to help us probe and analyze to get the character of a network (like requests and replies). ICMP messages are sent in IP data grams and at first thought you might think ICMP uses IP as a higher-level protocol for handling. In fact, ICMP is integrated with IP internally. Let's take a look at the ICMP data gram and break it down so we have a basis on how it is structured, this will help us in our discussions of how ICMP is used in network attacks.

IP Header



ICMP Header



Above we see the IP and ICMP headers and what fields they would contain, let's now look at some sample ICMP dumps to get a better understanding of the headers and how we should look at the data grams.
16:53:30.378637 192.168.168.130 > 192.168.168.131: icmp: echo request (ttl 32, id 35844, len 60)

0x0000	4500 003c 8c04 0000 2001 3c66 c0a8 a882	E..<.....<f....
0x0010	c0a8 a883 0800 4a5c 0200 0100 **6162 6364**	......J\....abcd
0x0020	**6566 6768 696a 6b6c 6d6e 6f70 7172 7374**	efghijklmnopqrst

0x0030 **7576 7761 6263 6465 6667 6869** uvwabcdefghi

16:53:30.378869 192.168.168.131 > 192.168.168.130: icmp: echo reply (ttl 128, id 4181, len 60)
0x0000 4500 003c 1055 0000 8001 5815 c0a8 a883 E..<.U....X.....
0x0010 c0a8 a882 0000 525c 0200 0100 **6162 6364** ......R\....abcd
0x0020 **6566 6768 696a 6b6c 6d6e 6f70 7172 7374** efghijklmnopqrst
0x0030 **7576 7761 6263 6465 6667 6869** uvwabcdefghi

Here we see how the ICMP header is integrated with the IP header, this is an example of an echo request and an echo reply. Let's break this down:
First off we have the IP header indicated in blue. We see from the second nibble that the header is 4-32 bit words, or 20 bytes. Also, at the 9th Byte offset we see what the embedded protocol is, in this case it's "1" or ICMP. So lets now move to the ICMP header marked in red. Notice the first 2 Bytes in the first packet, we have a hex value of 0800. Now look at our ICMP header diagram above, we see the first Byte is the Type and the second Byte is the Code. In this case, a type of 08 and code of 00 indicates this is an echo request. Now look at the second packer, we see a Type of 00 and a code of 00, this indicates an echo reply. We will take a look at some other Type/Code combinations in other examples; you could also reference http://www.iana.org/assignments/icmp-parameters for a full list of combinations and meanings. At Byte offsets 4-7 within the header are the identifiers and the sequence number, notice they maintain the same value in each packet, 0200 and 0100. The sender of the echo request initializes an identifier used to keep track of multiple different requests and a sequence number if multiple requests are sent to the same destination- in this case we sent 4 requests. If we looked at the other requests and replies in this sequence you would notice the identifier would remain the same but the sequence number would increase by one. For example, if we see constant changes of the identifier from the same source it might be a crafted packet, or if we see the same identifier from the source over a period of time it could be using covert communication with the ICMP packet. Some exploit tools such as Stacheldraht, TFT, and Loki used this field to communicate with a compromised host (for example, Straceldraht used a static ICMP identifier value of 667. Similarly, TFN used a static value of 456). Next, we have the ICMP payload indicated in bold, this will vary depending on the host configuration.

What's going on with the dump below?

00:03:58.936296 192.168.168.131 > 68.35.172.6: icmp: echo request [ttl 1] (id 31331, len 92)
0x0000 4500 005c 7a63 0000 0101 e5e8 c0a8 a883 E..\zc.....
0x0010 4423 ac06 0800 dfff 0300 1500 0000 0000 D#.....
0x0020 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0030 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0040 0000 0000 0000 0000 0000 0000 0000 0000 .....

0x0050 0000 0000 0000 0000 0000 0000 ....

00:03:58.945987 10.169.24.1 > 192.168.168.131: icmp: time exceeded in-transit for 192.168.168.131 > 68.35.172.6: icmp: echo request [ttl 1] (id 31331, len 92, bad cksum ae72!) [tos 0xc0] (ttl 255, id 9040, len 56)
0x0000 45c0 0038 2350 0000 ff01 0bdf 0aa9 1801 E..8#P.....
0x0010 c0a8 a883 0b00 2c76 0000 0000 **4500 005c** ......,v....E..\
0x0020 **7a63 0000 0101 ae72 c0a8 a883 4423 ac06** zc.....r....D#..
0x0030 **0800 dfff 0300 1500** ....

The first packet should be familiar, it's just like the echo request example we already covered, except that the ICMP payload is padded with '0's'-- we are not expecting a reply from this packet. Instead, we set a TTL value within the IP header equal to '1' for a specific purpose, notice at the 8th Byte offset we have a value of 01 whereas in the above example we had a TTL of 32 (or 20 in hex). Remember that the TTL value is set to give the packet a time-to-live value depending on how many routers it passes. So for instance, if we have a TTL of 32 we would hope the packet reaches its destination before it passes through 32 routers, otherwise the packet will be discarded and an error sent. What essentially happens when the router receives the packet with a TTL of 1 is it discards the echo request (like we expected) because it decremented the TTL value by '1', so now we have a null value for the TTL field and the packet stops being routed. Instead, the router now sends an ICMP error message back to the source. Take a look at the second packet, after the 20 Byte IP header we see the ICMP header. If we look at the first 2 Bytes we see an ICMP Type of 0b (11 in decimal) and a Code of 00. This tells us the error returned is 'Time Exceeded', specifically 'Time to Live exceeded in Transit'. Although this is an error message, it's exactly what we were requesting. The source receiving the error message will now record the router that sent it, and we start the process all over again but this time with a TTL of 2 instead. What's happening? This is simply the tracert program recording the path through routers to our final destination.

Okay, let's look at another example:

17:50:48.065569 192.168.168.131 > 192.168.168.133: icmp: echo request (ttl 128, id 4589, len 60)
0x0000 4500 003c 11ed 0000 8001 38bd c0a8 a883 E..<......8.....
0x0010 c0a8 a885 0800 395c 0300 1100 **6162 6364** ......9\....abcd
0x0020 **6566 6768 696a 6b6c 6d6e 6f70 7172 7374** efghijklmnopqrst
0x0030 **7576 7761 6263 6465 6667 6869** uvwabcdefghi

17:50:48.142248 192.168.168.132 > 192.168.168.131: icmp: host 192.168.168.133 unreachable - admin prohibited filter for 192.168.168.131 > 192.168.168.133: icmp: echo request (ttl 113, id 4589, len 60, bad cksum 1047!) (ttl 237, id 57902, len 56)
0x0000 4500 0038 e22e 0000 ed01 085d c0a8 a884 E..8.......]....

0x0010 c0a8 a883 030d df0c 0000 0000 **4500 003c** .....E..<
0x0020 **11ed 0000 7101 1047 c0a8 a883 c0a8 a885** ....q..G.....
0x0030 **0800 395c 0300 1100** ..9\....

The first packet shows another ping request, this time to 192.168.168.133. Look what we get back though, not a reply but an unreachable message-- let's see what's going on. Take a look at the first 2 Bytes of the ICMP header again, this time we have a Type of 03 and a Code of 0d. If we looked this up at iana.org we see a Type of 3 is 'Destination Unreachable' and a Code of 0d (13 in Decimal) is 'Administratively prohibited'. Notice who sent the packet, it wasn't the destination system but 192.168.168.132--- hmmm, this is a router preventing us from accessing this subnet. Let's now analyze the payload within the second ICMP packet, this will be important in understanding ICMP error messages. We find that it contains an exact match of the original IP header and the first 8 Bytes of the original ICMP header, basically this tells the source what protocol the error message is about-- otherwise the source would not know why the ping failed.

One last example:

20:36:48.410786,192.168.168.131,192.168.168.133,37,udp,29
4500 001d 8ce8 0000 8011 c9e0 c0a8 a883
c0a8 a885 07d6 0025 0009 13da 0000 0000

20:36:48.410786,192.168.168.132,192.168.168.131,,,icmp,56
45c0 0038 a797 0000 fe01 3c77 c0a8 a884
c0a8 a883 0303 e11e 0000 0000 **4500 001d**
8ce8 0000 7e11 cbe0 c0a8 a883 c0a8 a885
0414 07d6 0025 0009 13da

Let's now look at some packets without TcpDump telling us what's going on in simple English. What we see from the first packet is an IP header with a length of 20 Bytes and UPD as the protocol used in this transmission (notice the 9th Byte offset is equal to 11, 17 in decimal, which tells us it's UDP-- reference www.iana.org for a complete list of protocol numbers). So what is this user doing? Let's look at the destination port within the UDP header, listed in green text. What we see is at the 3rd and 4th Byte the value of 0025 in hex, or 37 in decimal--- this is the Time service most likely and it looks like the source is requesting the time from 192.168.168.133. The second packet in our trace is not the requested time from the destination, but an ICMP error message returned from a router. Notice we have a normal 20 Byte IP header with the value of 01 in the protocol field-- this is ICMP. If we look at the ICMP header we see a Type of 03 and a Code of 03, this tells us it's a 'Destination Unreachable' message, specifically a 'Port Unreachable' error.

So what is all of this telling us-- quite a bit really. First, we have found out which IP is the router, second we found out that 192.168.168.133 exists, and third we know that it's not serving a service on port 37-- all with one packet.

So, do the packets shown in the 3 examples divulge a lot of information, you bet, and this will lead us to our first discussion of how we can take advantage of ICMP and exploit a network.

What Else Can We Use ICMP For? Remember MICE

What I'll try to cover is ICMP reconnaissance through scanning and mapping, and an emphasis on how we can use intrusion detection to actively and passively detect operating systems.

Reconnaissance Through ICMP

This is the first step in an attackers effort to gain information about a network. We've already touched on a few examples of how a user can gain valuable information about a network using echo requests (to see if a host is alive) and tracert (to get a layout of the network). Let's look at a few more techniques in general.

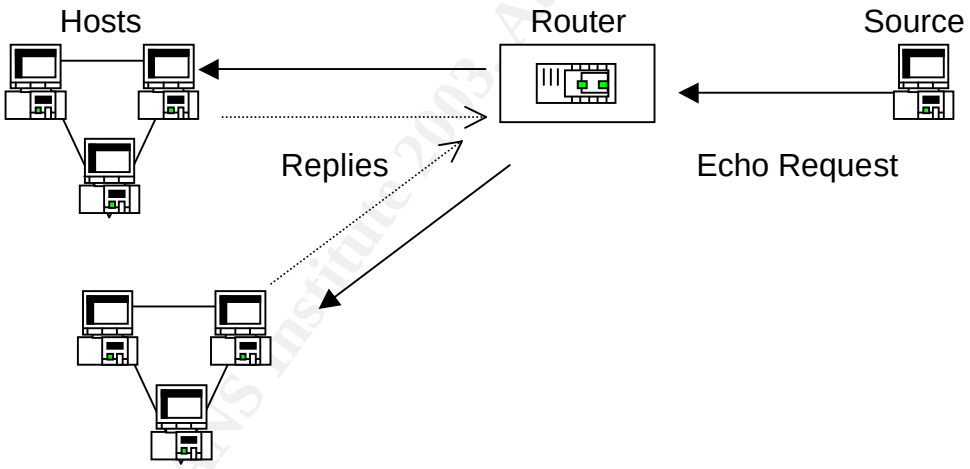


Fig 1. We see the source sending a single ICMP echo request to the broadcast address. The router forwards the requests to all internal hosts, then they respond with an echo reply.

Broadcasting: An efficient way to map and scan for live hosts is to send a single ICMP echo request to a router, but instead of the routers IP we send it to the broadcast address of the network. For example, in my network I use the 192.168.168.x network, with a subnet mask of 255.255.255.192. Essentially, this gives me 2 networks; one starting at 192.168.168.64 and the other at 192.168.168.128. With this configuration, my broadcast addresses are 192.168.168.127 and 192.168.168.191. If I were to ping one of the broadcast

address and my router accepted this type of ICMP (remember it's a Type of 08 and a Code of 00 for Ping) it would forward the request to every host on that network and they would all reply to the source--- well, not all the hosts. Windows systems do not respond to echo requests targeted toward a broadcast address, they silently discard the packets. So why would we try this technique in the present if we will not be able to accurately find all the hosts on a network? More often today this technique is used to find routers that accept a broadcast request in order to perform a denial of service on another network, or if enough systems respond it could create a DoS on their network-- more on this later. Another method of broadcast mapping is sending an echo request to the network address of the subnet. In my case the network address are 192.168.168.64 and 192.168.168.128. Again, Windows does not respond to this request either. Below are listed various OS's and whether or not they would answer and echo request aimed at the broadcast address they reside on.

Yes I would answer!
Linux
Solaris 2.5.1 +
Solaris 2.6 +
Solaris 2.7 +
Solaris 2.8 +
HP-UX v10.20 +
Routers, if enabled

No, go away!
All Windows Platforms
FreeBSD 4.0
FreeBSD 3.4
OpenBSD 2.7
OpenBSD 2.6
Routers, if properly configured

Address Mask Requests: As the name implies, this option will request the subnet mask of the network. Receiving the subnet mask can tell us a fair amount of information like determining live hosts, the network configuration and hierarchy, and potentially routers that would respond.

Before we go any farther let's briefly talk about a tool that seems to be the most popular for ICMP testing and manipulation, SING, available at http://www.sourceforge.net/projects/sing. Sing is used for manipulating certain fields in the ICMP header; this will solicit responses from the destination that will help us determine its operating system. SING let's us tailor ICMP packets to provide different scenarios we can examine, like fragmenting, spoofing, timestamps, router discovery, and address mask requests. It also let's us solicit error messages like Type 3 (unreachable) and Type 11 (time exceed). In our

case, we could use SING for every example that follows, it's that complete. Below is just one way to use this tool:

```
Melvin# sing -mask -c 1 192.168.168.135

192.168.168.134 > 192.168.168.135: icmp: address mask request (ttl 255, id
13170)
0000          4500 0020 3372 0000 FF01 FE99 c0a8 a886   E.. 3r.....:.
0010          c0a8 a887 1100 6BF7 8308 0000 0000 0000   ..:...k.....

192.168.168.135 > 192.168.168.134: icmp: address mask is 0xffffffC0 (ttl 60,
id 20187)
0000          4500 0020 4EDB 0000 3C01 A631 c0a8 a887   E.. N...<..1..:.
0010          c0a8 a886 1200 6BF6 8308 0000 FFFF FFC0   ..:...k.....
0020          0000 0000 0000 0000 0000 0000 0000        ..............
```

Again, let's take a look at these packets to see what we are requesting. In the first packet we see the source sends the router a Type 11 and Code 00 request--this translates to 'Address Mask Request', as TcpDump told us. The router responds in the second packet with a Type 12 and Code 00 reply, the 'Address Mask' answer. Notice in the payload we have a value of FFFFFFC0, or 255.255.255.192 in decimal. So now we can conclude that this network is subdivided into 2 networks at 62 hosts per network. So, with this simple request we not only get the configuration of the network but we also know a live router, and probably the main one for that network. Take a look at the list below, it shows what OS's would respond to this request, aside from routers.

NetBSD
Solaris 2.5.
Solaris 2.6
Solaris 2.7
Solaris 2.8
AIX v4.x
Windows
Windows 98
Windows 98 SE

Protocol Scanning: Here, we try to send IP packets with different protocol values, like 1 for ICMP, 6 for TCP, 17 for UDP and so on. Our intent is if the destination does not understand the protocol issued it will send us and ICMP Type 03 (Unreachable) Code 2 (protocol Unreachable) error message. If we do not receive an error message we are going to assume that protocol is understood, either that or a filtering device is in place silently dropping our packets. Let's take a look at how NMap can be used to scan a host for what protocols it understands. Notice how it sends raw IP packets, specifying a protocol but issues no protocol header at all or payload.

```
03:51:47.637064 192.168.168.130 > 192.168.168.131:  ip-proto-98 0 (ttl 48, id
7259, len 20)
0x0000          4500 0014 1c5b 0000 3062 9bd6 c0a8 a882    E....[..0b......
0x0010          c0a8 a883                                  ....

12:40:51.754556 192.168.168.131 > 192.168.168.130: icmp: 192.168.168.131
protocol 98 unreachable for 192.168.168.130 > 192.168.168.131:  ip-proto-98 0
(ttl 48, id 7259, len 20) (ttl 128, id 64941, len 56)
0x0000          4500 0038 fdad 0000 8001 6ac0 c0a8 a883    E..8......j.....
0x0010          c0a8 a882 0302 fcfd 0000 0000 4500 0014    ............E...
0x0020          1c5b 0000 3062 9bd6 c0a8 a882 c0a8 a883    .[..0b..........
0x0030          ffff ffff ffff ffff                        ........
```

The first packet we see shows an attempt to see if protocol 98 is understood on 192.168.168.131 (notice the 9th Byte offset has a hex value of 62, or 98 in decimal). Notice, also, we just have an IP header sent. The second packet we see tells us that the attempt failed, and issued a protocol 98 unreachable error message (Notice the ICMP Type and Code in red). Also, take a look inside the ICMP payload, we should have the original IP header and at least 8 Bytes of the protocol header in order for this packet to conform to normal transmit rules. Since we never sent a protocol header, the destination padded the missing 8 Bytes with all 'f's'.

Error Messages: We have seen how we can use Ping or broadcasts to map a network for live hosts, but we can also use ICMP error messages to accomplish the same thing. But, with these messages we can receive a lot more information than simple echo requests and replies. For instance, if we receive Type 3 error messages we can learn all kinds of information like host unreachable (probably host does not exist), port unreachable (port is closed), admin prohibited (to learn ACL's), and so on-- much better than a measly old reply packet.

Let's now turn to how we can start using these ICMP messages as a way to find out what OS's we have targeted.

Fingerprinting with ICMP

When we talk about fingerprinting NMap comes into most people's minds. But, using ICMP to fingerprint a host can give us the same information (if not more accurate) and give us a lot of other information as well, as we have seen. Also, using ICMP is less noisy and noticeable than trying half-open connections, or similar stealth techniques manipulating the flag bits. Another great attribute using ICMP is sometimes we can get all of this information with just one packet. We will now see that different operating systems deliver different ICMP messages, and by examining the fields we can determine what OS are sending them. Let's look at a very simple example, using the echo reply/request example from above.

16:53:30.378637 192.168.168.130 > 192.168.168.131: icmp: echo request (ttl 32, id 35844, len 60)

16:53:30.378869 192.168.168.131 > 192.168.168.130: icmp: echo reply (ttl 128, id 4181, len 60)

08:04:35.390092 192.168.168.130 > 192.168.168.131: icmp: echo request (ttl 32, id 36100, len 60)

16:53:31.387555 192.168.168.131 > 192.168.168.130: icmp: echo reply (ttl 128, id 4182, len 60)

Let's look at 2 particular parts of the packet, first the Time to Live field located at the 8th Byte offset of the IP header. We notice that the source sends a constant TTL of 32, and the destination sends a constant value of 128. This is indicative of Windows platforms, although the TTL of 128 could be a Netware system but we could initiate a protocol scan against it using NMap--- or, there are other fields we can look at if we solicit for other ICMP messages. Please reference http://project.honeynet.org/papers/finger/traces.txt for a list of default field values belonging to operating systems. We could also look at the IP ID numbers located at the 3rd Byte offset of the IP header. Basically, the IP ID value is used to keep track of fragments sent by a host, and the fragments will be reassembled if they contain the same value. So, if we initiate traffic that is not fragmented our IP ID should change accordingly. The OS can pick any value between 1 and 65,536 for the ID value, but each OS has a system for how it picks this so we might be able to find a pattern. In our example the source starts off with an ID of 35844, for the next echo request the ID changes to 36100 (a 256 difference). This is the behavior of older Windows platforms like Win 9x. The destination, however, follows a more standard approach for the IP ID, it increases it by '1' with every new connection.

This is a pretty easy example to analyze, especially since the dump is between 2 systems on the same subnet. So, what about systems we are analyzing that are out on the internet, will their TTL's and IP ID's be so easy to identify? No. Let's take an example: An echo request to a live host on the Internet.

15:44:33.319596 192.168.168.131 > 68.1.1.6: icmp: echo request (ttl 128, id 1362, len 60)
15:44:33.328818 68.1.1.6 > 192.168.168.131: icmp: echo reply (DF) (ttl 252, id 57007, len 60)

15:44:34.315226 192.168.168.131 > 68.1.1.6: icmp: echo request (ttl 128, id 1363, len 60)

15:44:34.323955 68.1.1.6 > 192.168.168.131: icmp: echo reply (DF) (ttl 252, id 57009, len 60)

Look at the TTL that the destination sends us, it has a value of 252. So, if we know the TTL's decrease by '1' with every router pass (see tracert example above) we also know this can't possibly be the initial TTL the OS selected (since the 68 network is outside of my 192 network). So what was the destination's original TTL value? One way to figure this out is with an educated guess using tracert back to the destination to calculate the number of hops. From me, it took 4 hops to get to 68.35.172.6. So if we add 4 to 252 we get 256 as a possible initial TTL. Although that does not seem to match any known signature, Cisco routers and Solaris systems are pretty close to this value as the initial TTL when sending an echo reply (they start off with a value of 255). Below is a list of default initial TTL values for OS's sending an echo reply: This may differ when the same OS sends an echo request however. Please refer to http://www.sys-security.com/html/projects/icmp.html for a pretty detailed and complete list of default field values for all the major OS's.

Linux Kernel 2.4	255
LinuxKernel 2.2.14	255
Linux Kernel 2.0.x47	64
Solaris 2.5.1	255
Solaris 2.6	255
Solaris 2.7	255
Solaris 2.8	255
Windows 95	32
Windows 98	128
Windows 98 SE	128
Windows ME	128
Windows NT	128
Windows 2000	128

Now look at the IP ID numbers coming from the destination, first it started with 57007 then ended with 57009. All this means is within that small fraction of time the destination processed 2 other connections before it responded to us with the echo reply (in other words the default behavior is not to increase the IP ID by 2 with every new connection).

Let's now take a look at another part of the same packets, the Don't Fragment (DF) field located at the 5th Byte offset of the IP header. We see our destination sent an ICMP reply with the DF flag set. Now we have a lot of information about this system; since it appears our destination has an initial TTL of 255, it increases the IP ID by '1', and it sets the DF flag on an echo reply we can be pretty sure this is a Solaris system. Notice the source did not set the DF bit, we too have a lot of information for this now; since it has an initial TTL of 128, it increases its IP ID by '1', and it does not set the DF bit when sending an echo request we

reiterate that this is a Windows box, in fact windows 2000. However, if we deliberately set the DF flag with other ICMP Types we might get back different results. For instance, if we sent an ICMP Address Mask Request with the DF flag set in the packet only Solaris and HPUX will echo back the DF flag, while Windows for instance will have a value set to 00 (no bits set). This is a great way to weed out OS's since it's pretty easy to distinguish Unix from Windows just by examining this field. Basically, I want to point out that the values we see in our ICMP messages may not be statically linked to OS's, but based on HOW we solicit the information **(***But, at any rate, each OS seems to have a signature for however we request data--- just not always the same value***)**.

Now we will look at yet another field that can clue us in on what OS is sending us the ICMP message. This time we will examine the Type of Service Byte, specifically the Type of Service (TOS) nibble located at the 0 Byte offset of the IP header. The TOS Byte gives IP some options to prioritize traffic, the TOS nibble deals with delays, reliability, and throughput. In our next example, we will set the TOS to be a value of 128, or 0x80 in hex. Some operating systems will handle this value being set differently, most will return a TOS with the same value while others will return their own TOS value. Let's see what happens when we send a Windows box an echo request with a TOS set at 128.

17:50:48.065569 ppp0 > 192.168.168.131 > 192.168.168.135: **icmp**: **echo request** [**tos 0x80**]
(ttl 255, id 13170)
45**80** 0024 3372 0000 ff01 d95d c0a8 a883
c0a8 a887 0800 df43 c304 0000 508c 0d3a
edf0 0a00

17:50:48.142248 ppp0 < 192.168.168.135 > 192.168.168.131: **icmp**: **echo reply** (ttl 111, id26133)
45**00** 0024 6615 0000 6f01 373b c0a8 a887
c0a8 a883 0000 e743 c304 0000 508c 0d3a
edf0 0a00

The Windows box returned an echo reply with a TOS set at '0', take a look at the 2nd Byte in bold, we have a value of 00. So, this is another way to try and nail down a particular OS, by weeding out most others. There are MANY variations and combinations we can use against the TOS field, especially in the unused and precedence bit of this field. Again, reference http://www.sys-ecurity.com/html/projects/icmp.html for a very detailed analysis of this.

We should now have a pretty good basis for common ICMP packets, both sent and received, and what fields we can concentrate on for analysis. We can then take this information and use it in reverse, to passively monitor our networks to determine what OS's are interested in us. Please note, we haven't even come close to a full analysis of all values and fields and bits we can manipulate with IP,

and all the variations of ICMP messages that would be generated-- to be fully confident with analyzing ICMP it will take a lot of practice, testing, generation, and time.

Passive Fingerprinting

We've covered some forms of active fingerprinting above, where we sent some form of a packet and waited for what kind of response we would get. Then we will analyze the fields and compare the values to known signatures. With passive fingerprinting, however, we either sit back and sniff the traffic as it comes to us (maybe from an attacker trying to do active fingerprinting on your system) or analyze the logs from a firewall or filtering device. Basically, we do essentially the same thing, we will analyze the packets against know signatures; the difference is we are not soliciting the requests. Some key advantages using this approach could be discovering systems otherwise protected by a filtering device, discovering unique programs someone could be using against you, revealing other entry points or configurations (like proxies or NATs), and it won't cause any additional load on the network like active techniques might.

What will be the key to passive fingerprinting with ICMP is concentrating on several issues. What we essentially want to know is what OS's send responses to certain stimuli, what OS's answer manipulated fields and how, and what kind of ICMP error messages will be sent. We will have a lot of different types of ICMP messages to analyze, and a lot of different fields to interpret. To recap, some key ICMP areas we should concentrate on when predicting an OS are the TOS Byte, the precedence bit, TTLs, DF flags, IP ID, ICMP sequence and identifier numbers, unused bits, and the payload.

In our next section, Network Detects, we will essentially analyze many of the same fields we did above, but this time for the TCP protocol. We will see that I tried to find anomalous data or values in these fields and also tried to predict what OS was sending traffic. I think the information we learned above will help us a lot when tackling the data dumps in our detects, in other words no matter what the protocol is we should take the same approach when analyzing.

Assignment # 2: Network Detects

Detect 1: WEB-IIS _vti_inf access & WEB-FRONTPAGE _vti_rpc access

1. Source of Trace.

The data file used came from http://www.incidents.org/logs/Raw/2002.6.13
The network layout appears to be a DMZ monitored by an IDS on the 46.5.0.0 subnet. The filtering device, whether a perimeter router or firewall, seems to be set up correctly for a DMZ allowing ports 80, 1080, 21, and 53 (probably a split DNS system) inbound, putting the burden of securing the subnet on the administrators governing the systems (i.e. service packs, patches, service/port auditing, wrappers, logging and so on). I'm not sure about the egress filtering but

it doesn't look like a lot of 46.5 addresses initiate connections outbound, just like servers shouldn't.

2. Detect was generated by: Snort IDS, Version 1.8.7beta5-ODBC-MySQL-MSSQL-WIN32 (Build 128) with **1.8.6 Ruleset**.

I ran the Raw data file using this command line: **snort -c snort.conf -l logs -r 2002.6.6 -vXyC**
The options used are as follows:
-c I used the default rule file snort.conf
-l is used to log the output from standard to the directory 'logs'
-r reads the input file I downloaded
-v verbose mode
-X I want the raw data, starting at the link layer
-y displays the year along with the timestamp
-C I wanted to see the payload interpreted (could have used **tcpshow** for this too)

The interesting detects I choose from the Snort alert file were this:

*** (Please note, I chopped the header information and similar log traces to save space, this will be my practice throughout the assignment since I took a lot of time and effort into analyzing these, and a lot of pages. I will provide the entire alert logs and corresponding TCPDump traces in separate files listed in the appendices). ***

[**] [1:990:5] WEB-IIS _vti_inf access [**]
[Classification: classtype] [Priority: 2]
07/12-18:34:49.644488 203.241.151.50:60592 -> 46.5.180.133:80

[**] [1:990:5] WEB-IIS _vti_inf access [**]
[Classification: classtype] [Priority: 2]
07/12-20:47:33.824488 218.17.234.225:38236 -> 46.5.180.133:80

[**] [1:990:5] WEB-IIS _vti_inf access [**]
[Classification: classtype] [Priority: 2]
07/13-11:34:47.744488 4.65.52.124:2114 -> 46.5.180.133:80

[**] [1:937:6] WEB-FRONTPAGE _vti_rpc access [**]
[Classification: classtype] [Priority: 2]
07/12-18:34:50.944488 203.241.151.50:61193 -> 46.5.180.133:80

[**] [1:937:6] WEB-FRONTPAGE _vti_rpc access [**]
[Classification: classtype] [Priority: 2]
07/12-20:47:34.644488 218.17.234.225:38237 -> 46.5.180.133:80

[**] [1:937:6] WEB-FRONTPAGE _vti_rpc access [**]

[Classification: classtype] [Priority: 2]
07/13-11:34:48.304488 4.65.52.124:2120 -> 46.5.180.133:80

Basically, we have 3 different source IP's connecting to a web server on 46.5.180.133, utilizing 2 different approaches (_vti_inf access attempts and _vti_rpc access attempts). We will see later that these 2 approaches really go hand-in-hand, and the server is hosting a web page with FrontPage extensions.

Let's break down what Snort is alerting us about, above the obvious WEB-FRONTPAGE warning message----> but what does that mean?

Look at the actual rule file below: I'll explain the most pertinent options below.

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS _vti_inf access";flags:A+; uricontent:"_vti_inf.html"; nocase; classtype:web-application-activity; sid:990; rev:5;)

First, this tells us Snort is set up to 'alert' us if these conditions apply--- what conditions--- continue on.....

Next, the protocol has to be tcp, we have tcp connections above so we are right on so far....

$EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS , what this means is if anyone outside of your homenet is going to any of the HTTP SERVERS defined on HTTP PORTS (in this case port 80), on any source port, the rule still applies.

Next, we have the message to display if the remaining variables are met, in this case the alert is titled: :"**WEB-IIS _vti_inf access**"

flags:A+ means we are looking for an ACK bit set and any other tcp flag set (denoted with the + sign. In our case we have ACKs and PSH set.

uricontent:"_vti_inf.html", this is telling us to look for that content anywhere in the payload, without attention to upper or lower case (nocase option).

The other rule file follows the same breakout, the rule is:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-FRONTPAGE _vti_rpc access"; flags:A+; uricontent:"/_vti_rpc"; nocase; reference:bugtraq,2144; classtype:web-application-activity; sid:937; rev:6;)
This one is nice, it gives us a bugtraq number, so we know it can be pretty nasty...

So, if we look at the dump below, we will see how it conforms to the alert message and it's variables (annotated in bold print).

*** Again, I will only provide enough dumps to help the narrative, the full dumps will be included as attachments in the appendicies. ***

windump -r e:\giac\raw\2002.6.13 -nXvv src 218.17.234.225 and dst 46.5.180.133

-r Reads the input file
-n Do not do nslookups on the IP's involved
-X Dump with character translation of payload
-vv verbose mode with displayed ttl's and IP ID's

18:34:49.644488 IP (tos 0x0, ttl 48, id 16760, len 314) 203.241.151.50.60592 > 46.5.180.133.**80**: P [bad tcp cksum 502d (->8047)!] **832713821:3832714083(262) ack 2125416356** win 65535 <nop,nop,timestamp 4461199 7368629>bad cksum 89e (->298)!

```
0x0000   4500 013a 4178 0000 3006 089e cbf1 9732      E..:Ax..0......2
0x0010   2e05 b485 ecb0 0050 e472 925d 7eaf 47a4      .......P.r.]~.G.
0x0020   8018 ffff 502d 0000 0101 080a 0044 128f      ....P-.......D..
0x0030   0070 6fb5 4745 5420 2f5f 7674 695f 696e      .po.GET./_vti_in
0x0040   662e 6874 6d6c 2048 5454 502f 312e 300d      f.html.HTTP/1.0.
0x0050   0a44 6174 653a 2053 6174 2c20 3133 204a      .Date:.Sat,.13.J
0x0060   756c 2032 3030 3220 3031 3a34 353a 3432      ul.2002.01:45:42
0x0070   2047 4d54 0d0a 4d49 4d45 2d56 6572 7369      .GMT..MIME-Versi
0x0080   6f6e 3a20 312e 300d 0a41 6363 6570 743a      on:.1.0..Accept:
0x0090   202a 2f2a 0d0a 5573 6572 2d41 6765 6e74      .*/*..User-Agent
0x00a0   3a20 4d6f 7a69 6c6c 612f 322e 3020 2863      :.Mozilla/2.0.(c
0x00b0   6f6d 7061 7469 626c 653b 204d 5320 4672      ompatible;.MS.Fr
0x00c0   6f6e 7450 6167 6520 342e 3029 0d0a 486f      ontPage.4.0)..Ho
<snip>
```

18:34:50.944488 IP (tos 0x0, ttl 48, id 24172, len 441) 203.241.151.50.61193 > 46.5.180.133.**80**: P [bad tcp cksum 9b37 (->b567)!] **390767304:3390767693(389) ack 2126797398** win 65535 <nop,nop,timestamp 4461202 7368758>bad cksum eb2a (->e524)!

```
0x0000   4500 01b9 5e6c 0000 3006 eb2a cbf1 9732      E...^l..0..*...2
0x0010   2e05 b485 ef09 0050 ca1b 00c8 7ec4 5a56      .......P....~.ZV
0x0020   8018 ffff 9b37 0000 0101 080a 0044 1292      .....7.......D..
0x0030   0070 7036 504f 5354 202f 5f76 7469 5f62      .pp6POST./_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 3435 3a34 3320      l.2002.01:45:43.
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:
<Snipped>
```

windump -r e:\giac\raw\2002.6.13 -nXvv src 218.17.234.225 and dst 46.5.180.133

```
20:47:33.824488 IP (tos 0x0, ttl 105, id 49675, len 280) 218.17.234.225.38236 >
46.5.180.133.80: P [bad tcp cksum b6e5 (->e6ff)!] 6700137:6700377(240) ack
1929180817 win 7788 (DF)bad cksum ad5c (->a756)!
0x0000   4500 0118 c20b 4000 6906 ad5c da11 eae1      E.....@.i..\....
0x0010   2e05 b485 955c 0050 0066 3c69 72fc f691      .....\.P.f<ir...
0x0020   5018 1e6c b6e5 0000 4745 5420 2f5f 7674      P..l....GET./_vt
0x0030   695f 696e 662e 6874 6d6c 2048 5454 502f      i_inf.html.HTTP/
0x0040   312e 310d 0a44 6174 653a 2053 6174 2c20      1.1..Date:.Sat,.
0x0050   3133 204a 756c 2032 3030 3220 3033 3a34      13.Jul.2002.03:4
0x0060   383a 3333 2047 4d54 0d0a 4d49 4d45 2d56      8:33.GMT..MIME-V
0x0070   6572 7369 6f6e 3a20 312e 300d 0a41 6363      ersion:.1.0..Acc
0x0080   6570 743a 202a 2f2a 0d0a 5573 6572 2d41      ept:.*/*..User-A
0x0090   6765 6e74 3a20 4d6f 7a69 6c6c 612f 322e      gent:.Mozilla/2.
0x00a0   3020 2863 6f6d 7061 7469 626c 653b 204d      0.(compatible;.M
0x00b0   5320 4672 6f6e 7450 6167 6520 342e 3029      S.FrontPage.4.0)
0x00c0   0d0a 486f 7374 3a20 7777 772e 5858 5858      ..Host:.www.XXXX
0x00d0   2e63 6f6d 0d0a 4163 6365 7074 3a20 6175      .com..Accept:.au
<snipped>

20:47:34.644488 IP (tos 0x0, ttl 105, id 55307, len 405) 218.17.234.225.38237 >
46.5.180.133.80: P [bad tcp cksum ae62 (->c892)!] 6701699:6702064(365) ack
1930933751 win 8484 (DF)bad cksum 96df (->90d9)!
0x0000   4500 0195 d80b 4000 6906 96df da11 eae1      E.....@.i.......
0x0010   2e05 b485 955d 0050 0066 4283 7317 b5f7      .....].P.fB.s...
0x0020   5018 2124 ae62 0000 504f 5354 202f 5f76      P.!$.b..POST./_v
0x0030   7469 5f62 696e 2f73 6874 6d6c 2e65 7865      ti_bin/shtml.exe
0x0040   2f5f 7674 695f 7270 6320 4854 5450 2f31      /_vti_rpc.HTTP/1
0x0050   2e31 0d0a 4461 7465 3a20 5361 742c 2031      .1..Date:.Sat,.1
0x0060   3320 4a75 6c20 3230 3032 2030 333a 3438      3.Jul.2002.03:48
0x0070   3a33 3320 474d 540d 0a4d 494d 452d 5665      :33.GMT..MIME-Ve
0x0080   7273 696f 6e3a 2031 2e30 0d0a 5573 6572      rsion:.1.0..User
0x0090   2d41 6765 6e74 3a20 4d53 4672 6f6e 7450      -Agent:.MSFrontP
0x00a0   6167 652f 342e 300d 0a48 6f73 743a 2077      age/4.0..Host:.w
0x00b0   7777 2e58 5858 582e 636f 6d0d 0a41 6363      ww.XXXX.com..Acc
0x00c0   6570 743a 2061 7574 682f 7369 6369 6c79      ept:.auth/sicily
0x00d0   0d0a 436f 6e74 656e 742d 4c65 6e67 7468      ..Content-Length
0x00e0   3a20 3431 0d0a 436f 6e74 656e 742d 5479      :.41..Content-Ty
```

<snippped>

windump -r e:\giac\raw\2002.6.13 -nXvv src 4.65.52.124 and dst 46.5.180.133

11:34:47.744488 IP (tos 0x0, ttl 108, id 23626, len 305) 4.65.52.124.2114 > 46.5
.180.133.**80**: P [bad tcp cksum 5d9 (->35f3)!] **4824952:4825217(265) ack**
2314464474 win 8064 (DF)bad cksum 9c3b (->9635)!

```
0x0000   4500 0131 5c4a 4000 6c06 9c3b 0441 347c      E..1\J@.l..;.A4|
0x0010   2e05 b485 0842 0050 0049 9f78 89f3 ecda      .....B.P.I.x....
0x0020   5018 1f80 05d9 0000 4745 5420 2f5f 7674      P.......GET./_vt
0x0030   695f 696e 662e 6874 6d6c 2048 5454 502f      i_inf.html.HTTP/
0x0040   312e 310d 0a44 6174 653a 2053 6174 2c20      1.1..Date:.Sat,.
0x0050   3133 204a 756c 2032 3030 3220 3138 3a33      13.Jul.2002.18:3
0x0060   343a 3033 2047 4d54 0d0a 4d49 4d45 2d56      4:03.GMT..MIME-V
0x0070   6572 7369 6f6e 3a20 312e 300d 0a41 6363      ersion:.1.0..Acc
0x0080   6570 743a 202a 2f2a 0d0a 5573 6572 2d41      ept:.*/*..User-A
0x0090   6765 6e74 3a20 4d6f 7a69 6c6c 612f 322e      gent:.Mozilla/2.
0x00a0   3020 2863 6f6d 7061 7469 626c 653b 204d      0.(compatible;.M
0x00b0   5320 4672 6f6e 7450 6167 6520 342e 3029      S.FrontPage.4.0)
0x00c0   0d0a 486f 7374 3a20 7777 772e 5858 5858      ..Host:.www.XXXX
0x00d0   2e63 6f6d 0d0a 4163 6365 7074 3a20 6175      .com..Accept:.au
```

<snipped>

11:34:48.304488 IP (tos 0x0, ttl 108, id 26442, len 430) 4.65.52.124.2120 > 46.5
.180.133.**80**: P [bad tcp cksum a581 (->bfb1)!] **4831330:4831720(390) ack**
2321404685 win 8760 (DF)bad cksum 90be (->8ab8)!

```
0x0000   4500 01ae 674a 4000 6c06 90be 0441 347c      E...gJ@.l....A4|
0x0010   2e05 b485 0848 0050 0049 b862 8a5d d30d      .....H.P.I.b.]..
0x0020   5018 2238 a581 0000 504f 5354 202f 5f76      P."8....POST./_v
0x0030   7469 5f62 696e 2f73 6874 6d6c 2e65 7865      ti_bin/shtml.exe
0x0040   2f5f 7674 695f 7270 6320 4854 5450 2f31      /_vti_rpc.HTTP/1
0x0050   2e31 0d0a 4461 7465 3a20 5361 742c 2031      .1..Date:.Sat,.1
0x0060   3320 4a75 6c20 3230 3032 2031 383a 3334      3.Jul.2002.18:34
0x0070   3a30 3420 474d 540d 0a4d 494d 452d 5665      :04.GMT..MIME-Ve
0x0080   7273 696f 6e3a 2031 2e30 0d0a 5573 6572      rsion:.1.0..User
0x0090   2d41 6765 6e74 3a20 4d53 4672 6f6e 7450      -Agent:.MSFrontP
0x00a0   6167 652f 342e 300d 0a48 6f73 743a 2077      age/4.0..Host:.w
0x00b0   7777 2e58 5858 582e 636f 6d0d 0a41 6363      ww.XXXX.com..Acc
0x00c0   6570 743a 2061 7574 682f 7369 6369 6c79      ept:.auth/sicily
0x00d0   0d0a 436f 6e74 656e 742d 4c65 6e67 7468      ..Content-Length
0x00e0   3a20 3431 0d0a 436f 6e74 656e 742d 5479      :.41..Content-Ty
0x00f0   7065 3a20 6170 706c 6963 6174 696f 6e2f      pe:.application/
0x0100   782d 7777 772d 666f 726d 2d75 726c 656e      x-www-form-urlen
0x0110   636f 6465 640d 0a58 2d56 6572 6d65 6572      coded..X-Vermeer
0x0120   2d43 6f6e 7465 6e74 2d54 7970 653a 2061      -Content-Type:.a
0x0130   7070 6c69 6361 7469 6f6e 2f78 2d77 7777      pplication/x-www
```

```
0x0140   2d66 6f72 6d2d 7572 6c65 6e63 6f64 6564        -form-urlencoded
0x0150   0d0a 436f 6e6e 6563 7469 6f6e 3a20 4b65        ..Connection:.Ke
0x0160   6570 2d41 6c69 7665 0d0a 4361 6368 652d        ep-Alive..Cache-
0x0170   436f 6e74 726f 6c3a 206e 6f2d 6361 6368        Control:.no-cach
0x0180   650d 0a0d 0a6d 6574 686f 643d 7365 7276        e....method=serv
0x0190   6572 2b76 6572 7369 6f6e 2533 6134 2532        er+version%3a4%2
0x01a0   6530 2532 6532 2532 6532 3631 310a             e0%2e2%2e2611.
```

do not pay attention to the IP and TCP checksum failures, this is do to the destinatin IP's being munged for study puposes

Okay, so we have some data that conforms to our Snort alert entry, but let's analyze it now.

3. Probability the source address was spoofed: I don't want to default to the "well there's sequence numbers and an established TCP connection denoted by the returned ACKs so they have to be valid source IPs", so I'll try to explain a bit further why I believe they are valid, and SEPARATE, IP's.

First off, whether this is reconnaissance or valid access attempts (which I believe it is and I'll explain later), the source IP's are actually requesting data with their first initial attempt at the _vti_inf directory. They then receive this data which is evident by their next request, _vti_rpc (I'll get to all of this in a minute). It would be useless to them if they were spoofing their IP, and even if they were sniffing the relies back to the spoofed host it's way too much trouble for the simple requests given, especially if these requests happen normally and legitimate all the time (what better way to get recon than with legitimate request....) And, because of the few attempts at requesting the data and the amount of data given back to the source IP's (I'll show this too), it's not enough to cause situations like a DoS to either bandwidth or CPU utilization.

Second, none of the IP or TCP fields seem crafted or spoofed (maybe I'm wrong here?), let's take a look at some particular fields: SYNs, ACKs, ports, TTLs, flags, window sizes, Header lengths, IP IDs, reserved fields, options, Time, and data.
SYNs: hey look, it appears we are already in the middle of an established connection to a web server.

IP: 203.241.151.50

We will see, in my opinion, that although the 2 different attempts to access certain directories go hand-n-hand, they initiate separate connections. Below are the sequence numbers, truly random in my opinion otherwise this system is flying around with all kinds of connections, grouped together by access attempts. What we see are separate sequence numbers for each attempt, just like separate connections should look like.

3832713821:3832714083(262)
3390767304:3390767693(389)

475005787:475006049(262)
1493364766:1493365155(389)

3203231433:3203231695(262)
3511070194:3511070583(389)

IP: 218.17.234.225

What we see here is possibly a random positive increment of sequence numbers for each attempt, otherwise we are missing some data here-----if the NEXT expected Seq number should be 6700377 but we have 6701699 then we should have seen 1322 bytes of data--- where is this? I don't think it exists because it's a new connection attempt, therefore the new seq number (if it's the exploit or valid attempt I'm thinking of).
6700137:6700377(240)
6701699:6702064(365)

IP: 4.65.52.124

4824952:4825217(265)
4831330:4831720(390)

Same goes for this too....otherwise we are missing 6113 bytes of data.

ACKs:

For IP 203.241.151.50
2125416356
2126797398-- looks like random positive, or the server returned 1.3 megs back to us, not likely for this request

2726606846
2720916284-- hmm, unusual here, the ACK decreased-- any ideas why, I'm not sure since it appears it normally increases???
2760327168-- same, random positive, otherwise server returned 8.7 megs!!!
2769305419

For IP 218.17.234.225
1929180817-- same, random positive
1930933751

For IP 4.65.52.124

2314464474-- same, random positive
2321404685

Ports: Okay, let's try to put to rest why I think these are separate connections. Here are the source ports for each connection, they change, and in mostly positive increments, just like new connections should.

IP 203.241.151.50:
60592
61193
60802
61384
15763
16446

Notice for each attempt the source port changes, this person's system is flying though, since within 10 minutes he already hit the max number of 65535, so it seems, and rolled over to lower number ephemeral ports. **AM I WRONG HERE?**

IP 218.17.234.225:
38236
38237

This system is doing much with this server at the moment, notice its ports only increased by 1.

IP 4.65.52.124:
2114
2120

Looks like a normal increase of ports.

TTLs: We have TTLs of **48, 105, and 108**. The 48 TTL belongs to 203.241.151.50 (probably initial TTL was 60). The 105 and 108 probably had initial TTLs of 128. Now, here's where we try to 'prove' this...

203.241.151.50 reports on ARIN.net: **OrgName: Asia Pacific Network Information Centre**

From me this took 20 hops---> 48 + 20 = 68, pretty close to predicted TTL.

218.17.234.225 reports: **descr: CHINANET Guangdong province network**

took me 23 hops----> 105 + 23 = 128, right on...

4.65.52.124 reports: **GTE Intelligent Network Services, US**

took me 17 hops----> 108 + 17 = 125, pretty close.

Okay, these are believable TTLs in my opinion.

Flags: On the 13th byte offset of each dump, we have a hex value of 0x18, this means that the Ack flag and Push flags are both set--- good, no reserved settings and nothing unusual about that flag combination. Since the MTU is probably Ethernet and we are not sending or requesting near the max size for this media, we could continue to accept data without the PUSH flag being set, but we set this to ensure the server processes this up to the higher level protocol fields and gives us the information (answer) we requested.

Window Sizes: 203.241.151.50 has a window size remaining constant at **65535**, with a TTL of 60-- it's probably a Cisco box. Why would a Cisco box be trying to connect to a web server (?) ----> one option is it could be acting as a NAT for its internal users, just a thought.

218.17.234.225 has a window size of **7788** and **8484**, looks like it was able to process some data and increase its size in the next connection attempt. And its TTL was guessed at 128, looks like a Windows box.

4.65.52.124 has a window size of **8064** and **8760**, this too looks like a Windows box.

Header lengths: All Header lengths, both IP and TCP checked out, only 203.241.151.50 had TCP options.

IP IDs: Each grouping of connections were about 1 second apart, so in that time the IP ID's changed quite a bit, I guess this is indicative of pretty busy systems, but I can't see how that's enough to go on to suspect crafting, especially since all the fields so far seem normal to me.

IP 203.241.151.50
16760
24172
11185-- this grouping was about 10 minutes after the first, so it seems the system hit the max number and rolled over.
21014
34783
43976

IP 218.17.234.225
49675
55307

IP 4.65.52.124
23626
26442

Reserved fields: No reserved fields in either the IP or TCP header were found in any of the packets

IP and TCP options: Only 203.241.151.50 had TCP options, utilizing the **nop,nop,timestamp** options. The option, length and value reported match the hex values as expected and are legitimate options... so I think these are okay too otherwise the destination host would reject anyway. There were no IP options at all, so a clever spoof like utilizing source routing could be set into this field but wasn't.

Time:

203.241.151.50 first came in on 18:34:49.644488 and had several access attempts within roughly 10 minutes

218.17.234.225 came in on 20:47:33.824488 and stayed about 1 second.

4.65.52.124 came in on 11:34:47.744488 and stayed about 1 second also.

Now, with this said it's very unlikely this is one valid IP surrounded by 2 other decoys (to mask real identify)--- let's play the odds, I don't think someone would space out the attempts like this. Second, if all three were spoofed it doesn't really accomplish anything, and if 2 were spoofed to try and 'mix in' with a real IP requesting the same data there doesn't seem to be enough "other" traffic to confuse things. So, my conclusion that these are real and separate IP's still stands.

Data: The payload for each IP looks like the normal response for each request, from perhaps the same type of browser too (refer to the payload data reported that's requested, it's the same for every connection... even the cisco reports the same, putting more belief that it's just NATing the real windows host behind it and not proxying--- again, just a thought). We'll look at valid payload later.

I put in a Dshield database search for the source IPs to see if they have been involved in other attacks and 203.241.151.50 and 4.65.52.124 had zero reports, 218.17.234.225 had 188 all SYN attempts to port 80---- I'm not sure that tells me too much, although it's a high number.

4. Description of the Attack. This is where it gets interesting, and left up to interpretation. And, without all the data from and to the IP's I can only speculate on what's happening. Further, these connections will be shown that they can look exactly the same, whether using them as reconnaissance or legitimate requests. There are many instances of exploits using the _vti_inf and _vti_rpc access. Let's look at a few since the data present does not fully conform to any of the known exploits, but does meet at least part of all of them. In our case, the request each IP's issued, POST./_vti_bin/**shtml.exe**/_vti_rpc.HTTP/1.1, has been used for numerous exploits and information gathering. Basically, the shtml.exe file is the browse binary, available to everyone, and is used to request the version of extensions available and IIS version, this is used after we first

requested if extensions were installed (GET ./_vti_inf.html.HTTP/). Let's take a look:

http://www.somewhere.com/_vti_inf.html ----> now, view the source:

<snipped>

<body>

<!-- _vti_inf.html version 0.100>

<!--

<snipped>

--><!-- FrontPage Configuration Information

FPVersion="4.0.2.5526"

FPShtmlScriptUrl="_vti_bin/shtml.dll/_vti_rpc"

FPAuthorScriptUrl="_vti_bin/_vti_aut/author.dll"

FPAdminScriptUrl="_vti_bin/_vti_adm/admin.dll"

<snipped>

There is some good info here, but what we are looking for is the FPShtmlScriptUrl="_vti_bin/shtml.dll/_vti_rpc return, now we know there are extensions installed and how to query them.... let's do that now...

POST /_vti_bin/shtml.dll/_vti_rpc HTTP/1.1.

The body of the POST request contains the command in the form "method=command", where "command" is a string indicating the operation that the FrontPage client wants the server to execute. In our case all the IPs issued a **method= server version** attempt (look at the payload dumps above, near the end). Here is a sample output the web server would have returned if the file existed, if it existed in the default directory we are calling, and if we still have "everyone" rights to this file:

HTTP/1.0 200 OK

Server: Microsoft-IIS/3.0

Date: Fri, 17 Apr 1998 02:04:43 GMT

Content-type: application/x-vermeer-rpc

<html><head><title>RPC packet</title></head>

<body>

<p>method=server version

<p>server version= 3.0.2.1706

major ver=3

minor ver=0

phase ver=2

ver incr=926

 <

/body>

</html>

Notice all the good info we have now, both system and extension versions.... yes, this can be used for recon for a later attack! Now that we know shtml exists and is responding, we are free to utilize any of the below exploits to see if they work.

CAN-2000-0413 :The shtml.exe program in the FrontPage extensions package of IIS 4.0 and 5.0 allows remote attackers to determine the physical path of HTML, HTM, ASP, and SHTML files by requesting a file that does not exist, which generates an error message that reveals the path.

Example: If we use these device names, MAILSLOT, PIPE, and UNC, appended to our URL we can get the physical path to the server if the file does not exist. We would receive the following error: Cannot open "C:\InetPub\wwwroot\pipe.htm": no such file or folder.

http://yourwebsite/_vti_bin/shtml.exe/pipe.htm

CAN-2000-0709 :Also it can be used as a denial of service attack that will disable all FrontPage operations on a web site. It does this by requesting a URL that includes a DOS device name, the server extensions will hang and will not service any further requests.

To use this as a denial of service attack, we would request a URL through the shtml.exe component of the FrontPage Server Extensions and use a DOS device name, but with an .htm extension.... some examples:

http://yourwebsite/_vti_bin/shtml.exe//com1.htm>
http://yourwebsite/_vti_bin/shtml.exe//prn.htm>
http://yourwebsite/_vti_bin/shtml.exe//aux.htm>

CAN-2000-0710 : This one is very similar to CAN-2000-0413

Whisker Tool : This is just one of many web vulnerability scanners. It uses perl database files to quickly scan for all kinds of neat things, including _vti_inf access and _vti_rpc access using shtml. Although I don't think this tool was used because Snort did not alert on any other web gathering techniques Whisker would have tried (I guess it could have been in evasive mode but why did Snort catch any of it then?) but it would be pretty trivial to modify the perl script to JUST scan for this, or make up your own (but as we will see later, that although the access attempts by the IP do yield some good info it falls short of any exploit we will talk about--- so what was the point of the recon if in fact that's what it was?)

Could it be legitimate web postings or FrontPage clients acting as Internet Browsers? (I think this may be the case, more on this later)

Let's now look at the data and try to figure out what's going on, and what the 'attacker' could do next---> **but why I feel it was not an attack**.

5. Attack Mechanism: What do we do after we have identified extensions are installed and are able to be queried for versions?

On Windows NT and IIS, FrontPage security is basically controlled by the access rights to the three files: Admin.dll, Author.dll, and Shtml.dll. These rights respectively determine administration, authoring, and browsing rights. For example, if a remote user is able to read and execute Author.dll, then that user is able to administrate the web site.

We could now send a POST to /_vti_bin/_vti_aut/author.dll, which is the authoring binary. The post data is **method=open+service**%3a3%2e0%2e2%2e1706&service%5fname=%2f (notice how it now uses the server's version, **3.0.2.1706**, from above recon). If the ACL of author.dll permits this request, the server responds with its server settings.

POST /_vti_bin/_vti_aut/author.dll HTTP/1.1

MIME-Version: 1.0

User-Agent: MSFrontPage/4.0

Accept: auth/sicily

Content-Length: 241

Content-Type: application/x-www-form-urlencoded

X-Vermeer-Content-Type: application/x-www-form-urlencoded

Connection: Keep-Alive

method=list+documents%3a3%2e0%2e2%2e1706&service%5fname=&listHidde
nDocs=false&listExplorerDocs=false&listRecurse=false&listFiles=true&listFolders
=true&listLinkI
nfo=false&listIncludeParent=true&listDerivedT=false&listBorders=false&initialUrl
=

Now we could change some settings like this: listHiddenDocs=True and listExplorerDocs=True and listLinkInfo=True and listIncludeParent=true.

To retrieve a file, you send this as the POST data: method=get+document%3a3%2e0%2e2%2e1105&service%5fname=&document%5fname=about%2fdefault%2ehtm&old%5ftheme%5fhtml=false&force=true&get%5foption=none&doc. We could just as well put files there too-- you get the point.

cool-- but how about this instead....**defacing** a web page.

<snipped>

ascta014p151.onda.com.br - - [12/Jul/2001:02:47:05 -0500] "GET **/_vti_inf.html** HTTP/1.0" 200 1716 "-" "Mozilla/2.0 (compatible; MS FrontPage 3.0)"
ascta014p151.onda.com.br - - [12/Jul/2001:02:47:06 -0500] "POST **/_vti_bin/_vti_aut/author.exe** HTTP/1.0" 200 252 "-" "MSFrontPage/3.0"
ascta014p151.onda.com.br - - [12/Jul/2001:02:47:11 -0500] "POST

/**_vti_bin/shtml.exe/_vti_rpc** HTTP/1.0" 200 227 "-" "MSFrontPage/3.0"
ascta014p151.onda.com.br - - [12/Jul/2001:02:47:19 -0500] "POST
/_vti_bin/_vti_aut/author.exe HTTP/1.0" 200 764 "-" "MSFrontPage/3.0"
ascta014p151.onda.com.br - - [12/Jul/2001:02:47:24 -0500] "POST

<snipped>

/_vti_bin/_vti_aut/author.exe HTTP/1.0" 200 296 "-" "MSFrontPage/3.0"
ascta014p151.onda.com.br - - [12/Jul/2001:02:53:11 -0500] "GET /**rbteam1.jpg**
HTTP/1.0" 200 55927 "-" "Mozilla/2.0 (compatible; MS FrontPage 3.0)"
ascta014p151.onda.com.br - - [12/Jul/2001:02:53:18 -0500] "POST
/_vti_bin/_vti_aut/author.exe HTTP/1.0" 200 297 "-" "MSFrontPage/3.0"

This person messed around with the server settings, like we discussed above when accessing the author.exe file, until he came across a jpg called rbteam, he was able to modify this file and repost it to the website.....

The **Whisker tool** can be used to scan for this as well; below is part of the source code that I thought was pertinent to our discussion, you can see the my comments, in ***bold italics.***

scan () / >> _**vti_inf.html** ***#look here, that's what we did above to discover if extensions were installed***

ifexist

set frontpage=1

array FPDirs = _private, _**vti_bin**, _vti_pvt, _vti_log, _vti_txt, _vti_cnf ***#We've seen this directory above***

<snipped>

scan () / >> _vti_bin/shtml.dll,_vti_bin/**shtml.exe** ***#Look at this, remember this will tell us the versions***

scan () _vti_pvt >> writeto.cnf,svcacl.cnf,services.cnf,access.cnf

scan () _private >> registrations.txt,register.txt,orders.txt,form_results.txt

<snipped>

usemeth **POST** ***#Notice how Whisker wants to perform the POST method for everything, but as we saw in our dump the IP's used a GET when requesting the _vti_inf file, this shows me Whisker is probably not used, although you can change the procedure method at the command line and use the GET method instead***

<snipped>

If iis == 1

scan () / >> _vti_bin/_vti_aut/author.dll

ifexist

print - We seem to have authoring access to the FrontPage web
#***Indeed, we do have this as illustrated in my example above***

<snipped>

scan () / >> _vti_bin/_vti_aut/author.exe #***Depends on what version of extensions are installed whether it is an .exe or .dll file***

<snipped>

These can be used to learn more about the server ***however, the default Snort rules look for these attempts, another reason why I feel Whisker was not used, and why I feel the "attackers" were not really doing anything malicious.***

scan () _vti_pvt >> access.cnf

info - Contains HTTP server-specific access control information

scan () _vti_pvt >> service.cnf

info - Contains meta-information about the web

scan () _vti_pvt >> services.cnf

info Contains the list of subwebs.

scan () _vti_pvt >> writeto.cnf

info Contains information about form handler result files

scan () _vti_pvt >> svcacl.cnf

<snipped>

Like I stated above in my in-line comments, Snort is already set up to flag on attempted access to _vti_bin directories and the info gathering directories of _vti_pvt (reference the **web-frontpage.rules** file in your Snort directory). So, what are these IP's doing then--- since it does not appear they are probing around further to see if they have admin or author rights, or trying to get any other recon information, it would either be a recon for limited information (**versions**) but would require additional probing at some time, or it's legitimate traffic.

Was this destination IP targeted at some point? I looked at 10 days of previous detects and I could not find any recon or additional data from the source IP's. I have noticed that our web server is the brunt of just about every attack out there though (*So, maybe it's already hacked and we have 3 rival hackers rushing around defacing or changing previous settings the hacker before him/her had changed -- if that's the case, why isn't Snort detecting this and only the shtml access?*) . So, I don't feel this web server was target by these 3 source IP's (everyone else in the world knows about it though, maybe it's hooked up with a google search or something so hence the lack of a port scan attempt). Further, we already know it has vulnerabilities by previous attack attempts by all kinds of IP's and the fact it appears to respond to our FrontPage extension requests (which is vulnerable enough-- see CVE's above), so all traffic to this server should be analyzed and suspect....

Was this a Stimulus or Response? This can be highly assumed it's a stimulus

to our web server. We are asking it if it has FrontPage extensions installed (for whatever reason, to exploit, recon, or publish -- whether defacement or legitimate). When we get the answer back we issue another attempt, this time to determine the version of the server and extension (again, for whatever reason like recon, known vulnerabilities, or to determine how to interact correctly with the server). In my opinion though, it falls short of being a valuable recon or exploit. Why----> they didn't probe any farther. All they know right now is the server version and extension versions, but unless they plan on testing the admins attention to service packs and patches by launching a DoS at a later time, they really didn't gain much info here. It's not an exploit since we don't see any other data from these source IP's that would match our Snort rule sets--- like if they tried to exploit our directory structure or inject device commands in our URL (see above, I already explained these). Again, they could be using an evasive mode with Whisker for instance to circumvent our IDS, but why didn't they use it for their initial recon that Snort did capture?---> this is also why I don't feel it's an attack.

6. Correlations: I have already provided some previous traces above that could correlate to our dumped data we've captured, and CVE references. But below I have provided some further links, and there are a bunch on the topic for similar techniques and data logged.

http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=shtml.exe

http://packetstormsecurity.nl/9910-exploits/webfolders.txt --- how servers respond to requests

http://lists.jammed.com/incidents/2001/07/0067.html --- the defacement example

http://archives.neohapsis.com/archives/vuln-dev/2000-q2/0896.html --- logs

http://www.ecos.de/~mailarc/embperl/2001-05/msg00068.html --- more logs

http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=shtml --- additional CVE resources that involve shtml.

Dshield reports:
203.241.151.50 zero reports
4.65.52.124 zero reports
218.17.234.225 had 188 all SYN attempts to port 80---- doesn't match this type of activity, not sure what this IP is going.

7. Evidence of active targeting: Well, after I analyzed the past 10 days of detects and did not notice the 3 IP's attempting any recon or any data really to this web server, I would assume that if they were indeed doing recon they did it at some other time (and have shown some patience since). More likely they probably did a google search of some sort and this web page was returned in their query. Of course the source IPs are targeting a specific host, but I don't feel it's malicious in nature (we will see why in the **Severity** section). Again, I didn't see a probe or general scan for this subnet by the IPs or any recon targeted toward it. It's not likely its a wrong number either since we are capturing data in the middle of an established tcp connection, *and since the second packets in the*

group (querying the version) is only initiated if you received valid confirmation the extensions exist (the _vti_inf file) it shows us the source IP's received the response. *****important*****

8. Severity:

Severity = (criticality + lethality) - (system countermeasures + network countermeasures) rank with, 1 (lowest) 5 (highest)

Criticality: how critical the target system is: This appears to be a DMZ web server and the perimeter device (router or firewall or both) seem to allow worldwide access to the site on this port. I'm going to assume it is not serving anything critical like personal information or e-commerce applications. So lets give it a **2**.

Lethality: measure how severe the damage is to the target if attack succeeds: In this case, if the attackers were going to perform a DoS and try to gain author rights to the web server (which is not evident from the dumps or Snort alerts) I would have given it a 4 since a DoS knocks the system offline and by gaining author rights you could potentially get system access as the system account running the web server (which is probably pretty powerful by default). For instance, you could remotely access at least a portion of your file system, including read, write, and execute permissions, it can be done through proxy servers to more easily disguise the originating IP address, passwords are often stored in global.asa and other files which may be used to attack other servers, and FrontPage has executable access to many system dll's including msvcrt40.dll, netapi32.dll, rpcltcl.dll, samlib.dll, and wsock32.dll, and so on....But in our case I will give this a **2**.

It appears like these users have used some form of FrontPage editor or explorer, visual interdev, or Office2000 products to either open a web page or analyze it (maybe it's a cool web page!)... maybe still, this web server is hosting some form of Office product as html sources and as the user attempts to open it by default they will attempt to utilize the FrontPage extensions to view it (funny nature of MS but I've seen it before--- comments?). All of these applications utilize the RPC call to use FrontPage extensions when attempting to access data that would call on these extensions (duh!) Normally, we would only see this type of data if someone was trying to publish from a FrontPage client to a FrontPage server using webfolders: used to give Office and FrontPage users the ability to publish and work with web content. The concept is that a web site becomes a part of Windows Explorer so that you can work with web content as if it were located locally or on a network drive.... or for exploits. But we don't see any evidence to support either claim, so I'm guessing it's a **false positive** flagged because they use the same extensions Snort is set up to alarm on.

System Countermeasures: strength measures of the target itself: Well, I've alluded that the burden is on the admin to ensure service packs, patches, port/service auditing, logging, and so on is accomplish since our perimeter device is not filtering this service for us. I can't provide concrete evidence to how well the system is maintained, but since it does have extensions enabled and allowed for

query this is a potential access point for hackers. And since I've seen numerous other exploit attempts again this IP from numerous source IP's (and responses to them) it must be a hot target. So let's say it's only a **2.**

Network Countermeasure: Like I stated above, this perimeter is not filtering this service (HTTP) or filtering what IP's can access it (Asia was allowed--- hmmm), it may be filtering other ports or protocols but since this is a web server and it's allowed through the filtering device it really does not do us any good. There is an IDS close by, so if we monitor this closely we at least have some response to what's going on. Let's give this a **2** also.

So, **(2 + 2) - (2 + 2) = 0**

I would recommend close monitoring of the source IP's for future recon, probing, or attempted exploits. I would probably set up a tcpdump filter to flag on the sources to destination and log offline.

9. Defensive recommendation: First off, if you need the extensions present make sure those valuable files (shtml, admin.dll and author.dll) are tight and have the necessary security settings (not "everyone" in other words). Tighten up the account that runs the web server, make sure it can't roam freely throughout the system, and make sure your NTFS permissions are set correctly throughout the partitions file structure. Move that darn web server directory off the system partition! (obvious reasons here)--- create another partition and set it up there. Also, make sure the system is patched, apply hotfixes!, and keep current with service packs and bugtrac postings and the like. You could also do this in addition: From Microsoft.

This article explains how to configure FrontPage to use a virtual server with no root Web. This configuration allows you to have only subwebs for your virtual server with no root Web.

do the following:

Create the subwebs you want to reside on your virtual server. These subwebs should be one level deep and be configured to use unique permissions.

In the content directory for your virtual server, open the file _vti_inf.html in notepad and add the directive FPNoRootWeb="1" after the other FrontPage directives as shown below.

```
-- FrontPage Configuration Information

FPVersion="4.0.0.2128"
FPShtmlScriptUrl="_vti_bin/shtml.dll/_vti_rpc"
FPAuthorScriptUrl="_vti_bin/_vti_aut/author.dll"
FPAdminScriptUrl="_vti_bin/_vti_adm/admin.dll"
FPNoRootWeb="1"
```

It is also recommended that you edit your master copy of the _vti_inf.html as you did in step 2. Your master copy of the _vti_inf.html is located in c:\Program files\Common files\Microsoft Shared\Web Server

Extensions\4.0\Bin\ on Windows. For UNIX, the master copy resides in /Usr/Local/Frontpage/Version4.0/Bin/.

Remove all of the _vti directories in the content area for the old root Web except for _vti_pvt. This file needs to stay so that FrontPage can access the Frontpg.lck file.

Remove any directory mappings that remain that reference the deleted _vti directories for the root Web. This includes the mapping for _vti_bin for the root Web.

This isn't bad, much more secure, right?

Or you could just uninstall the extensions but if you or your users need it to publish it will now block all web authoring, web administration, WebFolders, InterDev, and WebBot operations.... security over production I guess.

But if you're using IIS WITH FrontPage extensions, it's just a matter of time!!!!

If this is a web server hosting the Internet, then we really don't have much of a choice with the perimeter devices, security falls upon the administrator.

10. Multiple choice test question: What mechanism below shows an attempt at possible recon efforts against the target system?

a. 2002-09-12 22:07:48 1.1.1.1- W3SVC1 FTPSERVER 2.2.2.2 80 GET /_vti_inf.html - 404 2 4184 155 HTTP/1.0
b. 2002-09-12 22:07:49 1.1.1.1- W3SVC1 FTPSERVER 2.2.2.2 80 POST /_vti_bin/_vti_aut/author.exe HTTP/1.0- 202 2 4184 155 HTTP/1.0
c. http://2.2.2.2/_vti_bin/shtml.exe/pipe.htm
d. scan () _vti_pvt >> access.cnf
e. All of the above

Answer: it's (e), All of the above

a. shows an attempt to see if extensions are installed
b. shows an attempt to see if the author file is accessable --- we see a 202 from the server so it is!
c. shows an attempt to induce the server to give us its directory structure by feeding it a bogus link
d. shows Whisker scanning a cnf file within the _vti_pvt folder for additional information

Detect 2: WEB-IIS view source via translate header

1. Source of Trace.

The data file used came from http://www.incidents.org/logs/Raw/2002.6.5. This network layout too appears to be a DMZ monitored by an IDS on the 46.5.0.0 subnet (obviously since I'm looking at a detect again to the web server

at 46.5.180.133). The filtering device is probably a perimeter router or firewall since it is allowing ports 80, 1080, 21, and 53 (probably a split DNS system) inbound, all normal type DMZ services, and I'm not seeing any unusual session connects or data being served. This setup puts the burden of securing the subnet on the administrators governing the systems (i.e. service packs, patches, service/port auditing, wrappers, logging and so on). Since I don't seem to see much traffic initiated outbound by the 46.5 servers I can assume there is some egress filtering or the servers are just not used to initiate outbound connections, just like they shouldn't (but hey, if they are compromised there is the possibility they would try to initiate outbound connections back to the attacker, is Snort set up to monitor OUTBOUND connections--- probably not--- just a thought).

2. Detect was generated by: Snort IDS, Version 1.8.7beta5-ODBC-MySQL-MSSQL-WIN32 (Build 128) with 1.8.6 Ruleset.

I ran the Raw data file using this command line: **snort -c snort.conf -l logs -r 2002.6.5 -vXyC**
See the first detect for an explanation of the options used. One important option I used was **-C**, I wanted to see the payload interpreted because these are web requests and the payload is very important here to determine malicious behavior or valid attempts generating false positives.

The interesting detects I choose from the Snort alert file was this: There are a total of seven alerts, but I'll only show the first one (see appendix for the full list).

```
[**] [1:1042:6] WEB-IIS view source via translate header [**]
[Classification: "/swc] [Priority: 2]
07/05/02-05:21:05.954488 213.224.83.110:24306 -> 46.5.180.133:80
TCP TTL:51 TOS:0x0 ID:11157 IpLen:20 DgmLen:313 DF
***AP*** Seq: 0xC3E66382  Ack: 0x18722F7C  Win: 0x4470  TcpLen: 32
TCP Options (3) => NOP NOP TS: 980230841 2606656
[Xref => http://www.whitehats.com/info/IDS305]
[Xref => http://www.securityfocus.com/bid/1578]
```

Okay, we have one IP (213.224.83.110) connecting to our web server and it initiates a request to view some source code for an asp script, instead of initiating the script. The source IP utilizes a special header attachment called 'Translate' to get the desired information (part of FrontPage)--- big deal, what's so important about viewing the source code anyway?---> We will look at why this can be a lethal attack.
Let's break down what Snort is alerting us about, aside from WEB-IIS view source via translate header----> this really isn't telling me much so far....

Look at the actual rule file below: We should already know what the fields mean from my explanation in the first detect.

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS view source via translate header"; flags:A+; content: "Translate|3a| F"; nocase; reference:arachnids,305; reference:bugtraq,1578; classtype:web-application-activity; sid:1042; rev:6;)

I do want to talk about the content field: "Translate|3a| F", this is telling us to look for that content anywhere in the payload. We are specifically looking for the ASCII payload of **"Translate|3a| F"** (3a in hex is a colon in ASCII, but the exploit can be used either way).

Our rule file is nice, it gives us a bugtraq number we can research, also the actual alert is even nicer, it gives us reference links for this exploit,
[Xref => http://www.whitehats.com/info/IDS305]
[Xref => http://www.securityfocus.com/bid/1578]

Let's look at the tcpdump below, we will see how it conforms to the alert message and it's variables. Notice we have what appears to be an established TCP connection already in progress evident by the returning acks, also notice inside the payload we have the 'translate:.f' request (along with some other fields I've highlighted) initiated with the ASCII version. Please note I've only included one dump, see the appendix for all seven dumps I analyzed.

windump -r \giac\raw\2002.6.5 -nXvv src 203.241.151.50 and dst 46.5.180.133

05:21:05.954488 213.224.83.110.24306 > 46.5.180.133.80: P [bad tcp cksum 301a!] 3286655874:3286656135(261) ack 410136444 win 17520 <nop,nop,timestamp 980230841
2606656> (DF) (ttl 51, id 11157, len 313, bad cksum 1557!)

```
0x0000   4500 0139 2b95 4000 3306 1557 d5e0 536e      E..9+.@.3..W..Sn
0x0010   2e05 b485 5ef2  0050 c3e6 6382 1872 2f7c     ....^..P..c..r/|
0x0020   8018 4470 a683 0000 0101 080a 3a6d 22b9     ..Dp.........:m".
0x0030   0027 c640 5052 4f50  4649 4e44 202f 6d61     .'.@PROPFIND./ma
0x0040   696e 2048 5454 502f  312e 310d 0a48 6f73     in.HTTP/1.1..Hos
0x0050   743a 2077 7777 2e58 5858 582e 636f 6d0d     t:.www.XXXX.com.
0x0060   0a43 6f6e  6e65 6374 696f  6e3a 206b 6565    .Connection:.kee
0x0070   702d 616c 6976 650d 0a44 6570 7468 3a20     p-alive..Depth:.
0x0080   300d 0a74 7261 6e73 6c61 7465 3a20 660d     0..translate:.f.
0x0090   0a55 7365 722d 4167 656e 743a 204d 6963     .User-Agent:.Mic
0x00a0   726f  736f  6674 2d57 6562 4441 562d 4d69     rosoft-WebDAV-Mi
0x00b0   6e69 5265 6469 722f  352e 312e 3236 3030     niRedir/5.1.2600
0x00c0   0d0a 436f  6e74 656e 742d 4c65 6e67 7468     ..Content-Length
0x00d0   3a20 300d 0a50 7261 676d 613a 206e 6f2d     :.0..Pragma:.no-
0x00e0   6361 6368 650d 0a58 2d46 6f72  7761 7264     cache..X-Forward
0x00f0   6564 2d46 6f72  3a20 3231 332e 3131 392e     ed-For:.213.119.
0x0100   3138 2e32 3337 0d0a 5669 613a 2031 2e30     18.237..Via:.1.0
0x0110   206e 6368 6173 7330 3220 284e 6574 4361     .nchass02.(NetCa
```

```
0x0120   6368 6520 4e65 7441 7070 2f35 2e32 2e31      che.NetApp/5.2.1
0x0130   5231 4434 290d 0a0d 0a                       R1D4)....
```

do not pay attention to the IP and TCP checksum failures, this is do to the destination IP's being munged for study purposes

Okay, let's analyze all of this now.

3. Probability the source address was spoofed: Like I stated in my first detect, I'm not going to default to saying it is a valid IP just because it appears we are in the middle of an already established connection, I want to try and show any crafting or abnormalities going on that would give us some help in accurately answering this question.

***Also, take a look at the payload, do you see an 'x-forwarded-for' entry, then an IP (213.119.18.237)?

Hmmm, so this is the real IP trying to connect to us, or is it? Let's talk about x-forwarding first since this throws a kink in our discussion (since we could have a valid IP from 213.224.83.110 but a spoofed IP of 213.119.18.237 being proxied). I'm not sure how many cache/proxy servers provide this service, but one very popular Unix* application is Squid-- and since we are most likely dealing with an Open BSD box this is probably a good guess (see the 'window size' discussion below). Although Squid proxies default to sending the x-forwarded-for header containing the true client side IP, this cannot really be used as a sure way to determine who really is wanting to connect to you. Since users can supply headers, the level of security provided by depending upon this header is extremely low. Basically, we shouldn't trust the X-Forwarded-For header at all--- why? Well, If someone knows how to spoof a cookie they probably know how to forge HTTP headers as well, and thus the X-Forwarded-For header appended to it. Further, there are a bunch of tools that can be used to scan web servers for all kinds of vulnerabilities, and append spoofed and forged HTTP headers. Take a look at the cum security toolkit (CST) v1.2.

The CST toolkit contains a script scanner and a bunch of database scripts (user editable of course). The sample databases contain over 350 possible vulnerability scripts and directories. Among its features are a scanner with 5 different Anti-IDS tactics (hex-values, double slashes, self-reference dirs, parameter hiding and session splicing (we might take a look at this method later)), and can send fake "X-Forwarded-For:", "Referer:" and "User-Agent:" headers to hide your scan even more. --- cool, YES!

According to the Squid FAQ there is an option to replace the IP address with the string **'unknown'**, to enhance client side security. We can however assume the IP address seen by the Web server is valid (which would be the proxy server), since that is where replies are sent, and while it is easy to forge the source IP

address, that doesn't do much good if the user wanted a reply back.... So, okay, let's assume the IP contained within the header is okay (since it is within the 213 subnet also).

First off, whether these are exploit attempts or valid requests (which I believe it is and I'll explain later), the source IP (the proxy) is actually requesting data with the translate command header. I'm not sure if they are receiving that data for their requests but it would be pointless to spoof their IP since this exploit is not a buffer overflow and cannot be used to get you a root shell to upload a backdoor or something (well, it can be used to get you root, but you would need the data back, more on this later). Also, it is too much trouble to spoof and sniff the replies back, you'd have to be dealing with a pretty sorry TCP stack to maintain all the ACKs going back and forth, and what a storm that would cause (but yes, you could use hunt or some other hijacking tool, and be spoofing the IP's and maintaining an established connection, but it doesn't seem like a probable way to hack.... plus, where is the recon? More on that later).

Second, none of the IP or TCP fields seem crafted or spoofed (again, any help on this interpretation is appreciated). So, let's take a look at the major fields: SYNs, ACKs, ports, TTLs, flags, window sizes, Header lengths, IP IDs, reserved fields, ,options, and data.

SYNs: Okay, indeed it appears we are already in the middle of an established connection to a web server.

IP: 213.224.83.110

We will see, in my opinion, that although this IP has 7 request attempts they are all valid and normal tcp traffic. Below are the sequence numbers, and they appeared to be part of the same connection (in other words they were not spawned from new connection attempts to the web server) at first glance. What we see here is that although the next expected SEQ numbers don't match it just means the source IP is pushing some data to the server that is not giving Snort a headache and alerting us. But, however, if we expect the next sequence number to be 3286656135 but we get **3309646438**, the source IP must have pushed about 22 megs to the server!---> in one second?--->and then pushed about 17 megs a second later---> so it appears maybe these are separate connection attempts for each translate request (I'd have to set up some FrontPage (FP) authoring tools and test this, or read up expected behavior from the FP program.)

3286655874:3286656135(261)
3309646438:3309646700(262)
3326744762:3326745023(261)
3336391854:3336392116(262)
3356871756:3356872002(246)
3371880369:3371880630(261)

3383555324:3383555586(262)

ACKs: Notice how these ACKs do not increment sequentially, this is most probably due to its TCP stack offering random SEQ numbers, otherwise what we should most likely expect if the above 7 requests were part of the same connection are the numbers increasing according to the data sent (no bouncing up and down in the order--- maybe the packets are out of order though, but if that were the case the server is responding with a lot of data back to the source--- no likely for these requests).

410136444
401045905
414667540
403732061
418006712
403137426
409751189

Ports: Looking at the source ports it becomes even more apparent that these are separate connections attempts. Below, we see the ports change in positive increments, just like new connections should. We also see because of the fairly large increase in port numbers that this system is busy, and probably has many connections open.

24306
24455
24582
24656
24814
24922
25015

TTLs: We have a TTL of 51, let's assume its initial TTL was either 60 or 64. Now, let's try to 'prove' this by doing a traceroute to the source IP...

ARIN reports:
inetnum: 213.224.52.0 - 213.224.83.255
netname: TELENET
descr: Telenet Operaties N.V.
country: BE
<snipped>

From me this took 16 hops---> 51 + 16 = 67, pretty close to predicted TTL of 64 instead of 60, so let's use 64. Because of this, to me this is a believable TTL.

What about that "real" source IP being proxied?

ARIN reports:
inetnum: 213.118.160.0 - 213.119.255.255
netname: TELENET
descr: Telenet Operaties N.V.
country: BE

From me it took 19 hops--- looks like that proxy server is about 3 hops away from the client, makes since because they are subnetted..... this also reasonably proves the IP address in the x-forwarded-for header is correct, otherwise this is a pretty sorry attempt to spoof an IP, especially since it's within the same network.

Flags: On the 13th byte offset of the TCP header for each dump we have a hex value of 0x18, this means that the Ack flag and Push flags are both set--- good, there is nothing unusual about this flag combination (Since the MTU is probably Ethernet and we don't send or request near the max size we don't really need to have the PUSH flag set, we do this to ensure the server processes this up to the higher level protocol fields and gives us the information (answer) we requested.) The value of 0x18 also shows us no reserved settings are set, good too.

Window Sizes: Our source IP has a window size remaining constant at 17520, is this unusual?--- not really, and with a guessed TTL of 64 this system appears to be an OpenBSD OS.

Header lengths: All Header lengths, both IP and TCP checked out, the source did have TCP options (notice the 0x80 at the 12th byte offset of the tcpheader, it shows us there are 32 bytes in the header instead of the default 20 bytes, therefore we have some options)

IP IDs: The source IP issued 7 requests for information in about 2.5 seconds, and we again see his system is pretty busy by looking at the IP IDs, which increment with every new connection...

11157
44437
62869
9110 --Look here, this system is busy because it hit the max number of 65535 and rolled over
52118
7575 --Here we see it rolled over again.
34967

Reserved fields: No reserved fields in either the IP or TCP header were found in any of the packets

IP and TCP options: We did have TCP options, utilizing the nop,nop,timestamp

options. The option, length and value reported match the hex values as expected and are legitimate options... if these fields were corrupt or manipulated the destination host would reject anyway. There were no IP options at all, so a clever spoof like utilizing source routing could be set into this field but wasn't.

Data: Here's where it gets a bit more in-depth. By nature, requesting source codes utilizing the translate option header is suspect enough, but the exploit using this normally has a GET command accompanying it--- we don't see any evidence of the source using a GET command to our web server (can this technique still be used by using the GET command first, then followed by the translate header in order to 'fake out' our annalist into believing it's normal FrontPage traffic, especially since our IDS is not alerting on GETs that are issued?--- we will see if this is possible later, maybe the CST toolkit can do it if anyone wants to test it out-- remember it can session splice).

*****ALSO, notice in the payload the WebDAV HTTP option of **'PROPFIND'** ?--- well, guess what, it has a vulnerability too.... check out the Snort alert rule for this (which was not logged, it's just an example):

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC webdav propfind access"; content:"<a\:propfind"; nocase; content:"xmlns\:a=\"DAV\">"; nocase; flags:A+; reference:bugtraq,1656; reference:cve,CVE-2000-0869; classtype:web-application-activity; sid:1079; rev:7;)

Okay, I'll have to prove or disprove this exploit too before we can even get on with the actual Snort alert of "view source via translate header"----> this detect is becoming quite detailed now.

WebDAV is an extension to the HTTP protocol that allows remote authoring and management of web content. DAV (Distributed Authoring and Versioning) is a set of extensions to web-servers that allow data to be written as well as read (see RFC 2518). One of the extensions is the PROPFIND command, which queries the server for information about a page or document, such as if somebody is currently editing the page. There is also a Depth header used with PROFIND requests: taking the values 0 to infinity-- in this case "give properties for a single resource only", or "give properties for all resources. In our dump the source is only requesting information for one particular page, note the depth=0.

Here is an example of a normal PROPFIND command:
PROPFIND /public/docs/myFile.doc HTTP/1.1
Content-Type: text/xml
Content-Length: XXX
Depth: 0
Translate: f
...

```
<?xml version="1.0"?>
<a:propfind xmlns:a="DAV:">
<a:prop><a:getcontenttype/></a:prop>
<a:prop><a:getcontentlength/></a:prop>
</a:propfind>
```

This looks basically like we have in the above dump-- so it looks legitimate so far...yes???

This is what a server would respond to the request:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: 310

<a:multistatus
  xmlns:b="urn:uuid:c2f41010-65b3-11d1-a29f-00aa00c14882/"
  xmlns:a="DAV:">
 <a:response>
   <a:href>http://server/public/test2/item1.txt</a:href>
   <a:propstat>
    <a:status>HTTP/1.1 200 OK</a:status>
      <a:prop>
       <a:getcontenttype>text/plain</a:getcontenttype>
       <a:getcontentlength b:dt="int">33</a:getcontentlength>
      </a:prop>
   </a:propstat>
 </a:response>
</a:multistatus>
```

Normal PROPFIND requests are only a few hundred bytes long (like we have) but an overflow DoS attack was discovered against IIS 5 DAV extensions whereby a large request could crash the server (see CAN-2001-0151 at cve.mitre.org). The attacker would not be able to control the WebDAV process or access any data just perform a DoS... so spoofing the IP would be beneficial in this case, but I think I've proven the IP is not spoofed and our data requested is within normal PROPFIND commands issued....

Here is the DoS exploit written in Perl: Notice the </a:propfind> within the code, Snort would have alerted us on this (see alert rule above), notice also it's trying to send way more than a couple hundred bytes (128008!) ---> see appendix.
So let's say this is not a DoS attempt either for a few reasons (Snort did not capture it, and the source IP only tried 7 request within a couple of seconds--- not much of a DoS in my opinion since the target system WILL regain control after the malformed requests stopped). So, let's also say the payload for each request looks like normal requests for web authoring, and by looking at the data size they all are around the 260 byte area, so this shows to me they are very similar

requests too. I also placed a Dshield database search for the source IP and "real" client IP to see if they have been involved in other attacks and recon. I'm still waiting for the database search to be emailed to me.

4. Description of the Attack: Just like my first detect, this is where it gets interesting and left up to interpretation--- without all the data from and to the IP's I can only assume what's happening. However, I'll try to show that even though a valid attempt for requesting the source code and an exploit attempt may look very similar, there are some key differences that we can hone in on.

Let's take a look at the translate:.f exploit and try to break it down and see what it is trying to accomplish. Then, we can compare it to our data dump above and notice they do not fully conform to one another. Although on the surface our Snort capture looks like an exploit, I'll try to show that our data falls short of any exploit utilizing this command header. In our case, the request that the source IP issued was translate:.f -- but what does this mean and what does it do?

Translate:.f is a Windows 2000 and IIS 5 'feature' that allows the source of ASP pages to be viewed through a simple HTTP request and is used by DAV-aware applications, like Office2K, IE5 or W2K's Web Folders (you can see my first detect about FrontPage extensions and web folders). Each client user can mount a WebDAV volume located on a shared server to his/her desktop. As long as the server supports WebDAV, users can mount the WebDAV volumes to their desktop regardless of the web server's operating system. Users can then access the files as if they were on any other networked volume. The exploit is used like this: Normally an HTTP header is not visible to the user. It contains information such as the type of character set the request uses, or proof that the user is authorized to view the requested page. The specialized header that tricks IIS into returning the unprocessed script source can't be created via a standard Internet browser, but it's easy for an attacker to create an HTTP request outside a browser and supply their own header information.

Here is an example of sending an HTTP header request using Netcat for retrieving the source code of default.asp on 1.1.1.1:

```
$ nc 1.1.1.1 80
GET /default.asp%5C HTTP/1.0
Host: 1.1.1.1
User-Agent: Mozilla/4.0
Content-Length: 18
Content-Type: text/html
Translate: f
match=www&errors=0
```

Note the use of **%5C** in the GET request, that's ASCII for the backslash character (\) and that's the key. So just by implementing this character, we can bypass all the security and tell the server to give us the code instead of the output.... hmmm

In other words, if an attacker requests a scriptable page, such as an ASP page, and adds Translate:f into the headers of the HTTP GET request and a backslash, an un-patched Windows 2000 machine will return the complete code instead of the processed file. Because DAV works via HTTP it uses the command GET to retrieve a file, but for script files this won't work as you'd just end up with the resulting output. So, MS came up with an extension where the DAV client sends a 'translate' header along too, which tells the server whether to 'translate' the requested file. For a script file, specifying translate:f in our case tells the server to NOT translate (ie process) the script but to just send back the source code. With this allowed, a client can request the source of any script-mapped files via a DAV client. If the 'Write' box was enabled too, they could upload a new version too--- you can see one possible motive for this now. By default, only Read is selected for websites anyway, so any user with a DAV client could only get the output of any file--- but we will see how this can be just as lethal later.

So, to summarize, Translate: f is a legitimate header for WebDAV, but by adding this to HTTP GET reveals the security hole, and asks the server to really return SOURCE code of file and bypass processing, and the backslash helps us bypass authentication... If we look at our tcpdump data we don't find any instance of the translate option used in conjunction with the GET command, but let's not dismiss this as a false positive yet until we go through the Attack Mechanism.

Reference
CVE CVE-2000-0778
Bugtraq 1578
Microsoft Security Bulletin (MS00-058)

5. Attack Mechanism: What can we do by requesting the source code of a page, and what methods are available to exploit this? So, why is viewing the source code a big deal. Well, a lot of sensitive information could be kept in these files, and programmers often rely on the fact that the source code is hidden from the user. Let's look at one particular exploit called srcgrab.pl: See appendix.

If we wanted to try and target a system, here is the command that would be used: **$ perl srcgrab.pl http://1.1.1.1/contact.asp.** In this case our IIS server was patched and did not honor the request: Notice the backslash after contact.asp and the 403 error.

2002-09-18 22:05:31 2.2.2.2 - W3SVC1 WEBSERV 1.1.1.1 80 GET /contact.asp\ - 403 0 179 211 HTTP/1.0 1.1.1.1 libwww-perl/5.64 -
2002-09-18 22:08:55 2.2.2.2 - W3SVC1 WEBSERV 1.1.1.1 80 GET /contact.asp\ - 403 0 179 211 HTTP/1.0 1.1.1.1 libwww-perl/5.64 -

Here is an example of what the dump looks like: notice how it differs from our tcpdump data by issuing the GET command, also notice after the requested page

'contact.asp' there is the backslash '\'. This is what we would expect if the source IP was indeed trying to take advantage of this exploit.

```
IIS-view source via translate\: F header
2002-09-18 22:08:55 attacker:15253 -> target:80
TCP TTL:53 TOS:0x0 ID:31083  DF
*****PA* Seq: 0x90F90192   Ack: 0x251D6AF2   Win: 0x2238
TCP Options => NOP NOP TS: 3060457 0
47 45 54 20 2F 70 61 67 65 73 2F 63 6F 6E 74 61    GET /pages/conta
63 74 2E 61 73 70 5C 20 48 54 54 50 2F 31 2E 30    ct.asp\ HTTP/1.0
0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 6C 69   ..User-Agent: li
62 77 77 77 2D 70 65 72 6C 2F 35 2E 34 35 0D 0A   bwww-perl/5.45..
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65   Content-Type: te
78 74 2F 68 74 6D 6C 0D 0A 54 72 61 6E 73 6C 61   xt/html..Transla
74 65 3A 20 66 0D 0A 58 2D 46 6F 72 77 61 72 64   te: f..X-Forward
65 64 2D 46 6F 72 3A 20 32 30 39 2E 31 38 37 2E   ed-For: 103.164.
31 34 30 2E 31 39 32 0D 0A 48 6F 73 74 3A 20 77   002.134..Host: w
<snipped>
```

But what are we really after here--- let's take a look if the exploit had succeeded.

```
Server: Microsoft-IIS/5.0
Date: Thursday, 18 September 2002 22:08:55 GMT
Content-Type: application/octet-stream
Content-Length:2200
ETag: "0448299fcd6bf1:bea"
Last Modified: Thu, 15 Aug 2002 17:25:41 GMT
Accept-Ranges: bytes
Cache-Control: no-cache
<!-Copyright 2000-2001 Webserv.com--->
<object RUNAT=Server SCOPE=Session ID=fixit
PROGID="webserv.object"></object>
("ConnectionText") = "PHone;UID=jsmith;Password=W3bMa$tr
("ConnectionText") = "PHone;UID=tjones;Password=s3rVrG0D
("LDAPServer") = "LDAP://ldap.websrv.com:389"
("LDAPUserID") = "Admin"
("LDAPPwd") = "LdapPa$$"
<snipped>
```

Look at all of this good information. We have several usernames and passwords now to work with, including the authors and database users. So, using tools such as netcat and srcgrab we can obtain the ASP source code for any page we desire. This becomes very dangerous when these ASP pages connect to databases, and as the example shows above we see the login and password for the database-- often the developer thinks the code will never be seen by the user and thus thinks it is safe to hardcode this information into the source. Even more

disturbing: SQL Server is often set up to share a local Windows login. In this case, the login and password for the database are the same as for the system, and a hacker can easily have full access to the system as if acting as that account.

But let's look at one of our tcp raw dumps from above: what we notice is we have already disproved the PROPFIND DoS and the probability that the x-forwarded-for IP is spoofed... so what's left? We can see the file being accessed (main), the host serving the page, our translate:.f header requesting the source for (main), and other normal HTTP options and parameters. So, I can safely assume that this is legitimate WebDAV traffic probably from a FrontPage client attempting to author and modify an ASP page---> why? There is no evidence of a GET command issued, or the tricky backslash, or any other manipulation that would be suspect (but, maybe it's already hacked and the attacker already has the passwords and is using those credentials to author in a now expected manner....???.... but, I just didn't see any evidence of this from past data files---so I'm playing the odds here).
Was this destination IP targeted at some point? I looked at 10 days of previous detects and I could not find any recon or additional data from the source IP. I have noticed, however, that our web server is constantly under attack from just about every attack out there. So, maybe it's hooked up with a google search (hence the lack of a port scan attempt) or it's a trusted server to the source IP. We know that the server either is serving up vulnerable services or has vulnerabilities from previous attacks by all kinds of IP's. So, we can probably assume it is responding to the translate options and thus running some form of WebDAV and FrontPage (which are vulnerable in their own rights). So, even though this appears to be legitimate traffic I would monitor this IP and continue to analyze and suspect all traffic.
Was this a Stimulus or Response? This can be highly assumed it's a stimulus to our web server. We are asking it first to give us file properties for an ASP page utilizing the PROPFIND command. Then we issued a translate:.f option to tell the server to not process the script but to let us view the source for modification and authoring. Whether this is legitimate traffic or an exploit we are soliciting the server for information.

6. Correlations: I have already provided sample traces from users above that could correlate to our dumped data we've captured, and CVE references. Below I also provide some links, and there are tons if you do a google search, to similar techniques and data logged.

http://www.whitehats.com/info/IDS305----> The IIS sample dump
http://www.securiteam.com/windowsntfocus/Translate_f_vulnerability_exposes_IIS_files_source.html---> exploit example
http://archives.neohapsis.com/archives/ntbugtraq/2000-q3/0080.html---> vulnerability and exploit information
http://www.learnasp.com/learn/translatef.asp----> test your vulnerability to the

exploit
http://www.4guysfromrolla.com/webtech/081500-1.shtml----> good info on what it's all about
http://www.advisor.com/Articles.nsf/aid/COBBM65---> more good info

Dshield reports: Still waiting on the results....

7. Evidence of active targeting: I analyzed the past 10 days of detects and did not notice the source IP attempting any recon or any data request to this web server, I would assume that if they were indeed doing an exploit they did their recon at some other time (and have shown patience since-- but we should have a Snort log of it somewhere). More likely they probably did a google search of some sort and this web page was returned in their query, or this is a known server to them. The source IP is targeting a specific host, but I don't feel it's malicious in nature (from our breakdown of the data in the sections above). Again, I didn't see a prob or general scan for this subnet by the IP or any recon targeted toward it. It's not likely its a wrong number either since we are capturing data in the middle of an established tcp connection, and since the IP is requesting data and the exploit is only used for data collection (not DoS or trojan or whatever) the IP will need the responses back ***important***

8. Severity:

Criticality: how critical the target system is: This appears to be a DMZ web server and the perimeter device (router or firewall or both) seem to allow world-wide access to the site on this port. I'm going to assume it is not serving anything critical like personal information or e-commerce aps. So lets give it a **2**.

Lethality: measure how severe the damage is to the target if attack succeeds: In this case, if the attacker was going to exploit the asp source code, or utilize a PROPFIND DoS attack (both of which are not evident in our dumps) I would have given this a 4 since a DoS knocks the system offline and by gaining the source code we could potentially gain usernames and passwords or act as the system account (like getting the global.asa page for instance). But in our case I will give this a **2**. It appears like these users have used some form of Frontpage editor or explorer, visual interdev, or Office2000 products to either open a web page and author it. So I'm guessing it's a false positive flagged because they use the same calls that Snort is alerting us on.

System Countermeasures: strength measures of the target itself: Well, I've alluded that the burden is on the admin to ensure service packs, patches, port/service auditing, logging, and so on is accomplish since our perimeter device is not filtering this service for us. I can't provide concrete evidence to how well the system is maintained, but since it does seem to have WebDAV components installed and thus some Frontpage components as well this is a potential access point for hackers, and indeed a prime target. And since I've seen numerous

other exploit attempts against this IP from numerous source IP's (and responses to them) it must be a hot target. So let's say this value is a **2** also.

Network Countermeasure: Like I stated above, this perimeter is not filtering this service (HTTP) or filtering what IP's can access it (this time a Be IP is allowed), it may be filtering other ports or protocols but since this is a web server and it's allowed through the filtering device it really does not do us any good. There is an IDS close by, so if we monitor this closely we at least have some response to what's going on. Let's give this a **2** also.

So, (2 + 2) - (2 + 2) = 0

I would recommend close monitoring of the source IP for future recon, probing, or attempted exploits--- but most likely I would monitor it for excessive web page authoring attempts and abnormalities to the web server. I would probably set up a tcpdump filter to flag on the source to destination and log offline.

9. Defensive recommendation: Couldn't we modify the Snort alert rule to not just search for a nocase option of "translate:.f but also "asp\"?? I think if we did this we could decrease the amount of false positives for this type of access attempt... the Snort rule could look something like this:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS view source via translate header"; flags:A+; \
content: "asp\";nocase;      \
content: "Translate|3a| F"; nocase; \
reference:arachnids,305; reference:bugtraq,1578; classtype:web-application-
activity; sid:1042; rev:6;)
```

We might want to do a rule with "content: "asp|5c|";nocase;" to express the backslash as ASCII too. Any suggestions???

Continuing, if you need this feature installed to support remote authoring, first off, Download the patch...
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=23769
Note: This vulnerability is eliminated by installing Windows 2000 Service Pack 1

Also, tighten up the account that runs the web server, make sure it can't roam freely throughout the system, and make sure your NTFS permissions are set correctly throughout the partitions file structure. Move the web server directory off the system partition! --- create another partition and set it up there. Also, make sure the system is patched, apply hotfixes!, and keep current with service packs and bugtrac postings and the like.

10. Multiple choice test question: What log(s) show(s) an attempt to exploit a server with WebDAV componets installed?

a. PROPFIND /public/docs/myFile.doc HTTP/1.1
Content-Type: text/xml
Content-Length: 262
Depth: 0
Translate: f
b. GET /pages/contact.asp\ HTTP/1.0
c. 2002-09-18 22:05:31 2.2.2.2 - W3SVC1 WEBSERV 1.1.1.1 80 GET /contact.asp\
d. X-Forwarded-For:. 1.1.1.1

Answer: it's (b) and (c)
a. shows an attempt to obtain file properties, although this is a stimulus to gain information it is a valid attempt --- notice the content size...
b. shows an attempt to exploit the ASP source code by utilizing the backslash after the requested page
c. shows essentially the same thing but it's an IIS log
d. shows the Squid Proxy's client IP address and has nothing to do with this exploit

Detect 3: WEB-MISC http directory traversal

1. Source of Trace.
The data file used came from http://www.incidents.org/logs/Raw/2002.6.7. The 46.5.0.0 subnet for this data appears to be a DMZ servicing web, socks, ftp and DNS applications. There appears to be a perimeter filtering device blocking other ports and services because I don't see any other connection or scan attempts to non-DMZ applications. Also, there appears to be egress filtering too since I don't see traffic initiated outbound by the 46.5 servers except for occasional 403 forbidden errors. Because I see a wide range of unique IP address coming into this subnet I am assuming the filtering device allows all external addresses into the DMZ for these services. This setup puts the burden of securing the subnet on the administrators governing the systems, and a larger emphasis on host logging becomes important.

2. Detect was generated by: Snort IDS, Version 1.8.7beta5-ODBC-MySQL-MSSQL-WIN32 (Build 128) with 1.8.6 Ruleset.

I ran the Raw data file using this command line: snort -c snort.conf -l logs -r 2002.6.7 -vXyC

The interesting detects I choose from the Snort alert file was this: Again, only one is shown to save space, all 16 can be viewed in the appendix.

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/07/02-14:16:47.264488 66.108.153.160:55804 -> 46.5.180.133:80

TCP TTL:48 TOS:0x0 ID:63896 IpLen:20 DgmLen:301 DF
AP Seq: 0xC2FC290F Ack: 0x7A4E298C Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 16830631 5820679
[Xref => http://www.whitehats.com/info/IDS297]

The Snort alerts show us one IP (66.108.153.160) connecting to our web server (46.5.180.133) trying an apparent directory traversal attack. This is where a web application may allow access to a particular portion of the fire system without properly checking the user input. In this instance the source IP places "../" in the URL path trying to access parent directories normally restricted to the user. Let's break down what Snort is alerting us about, aside from WEB-MISC http directory traversal, and discuss what triggered this alert.

Look at the actual rule file below: I'll explain some of the fields below.

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC http directory traversal"; flags:A+; content: "../"; reference:arachnids,297; classtype:attempted-recon; sid:1113; rev:4;)

Content: "**../**", this is telling us to look for that content anywhere in the payload. We are specifically looking for the ASCII payload of this. This will become the meat of our discussion.

Let's look at the tcpdump below, we will see how it helps us understand why the alert was triggered. Notice we have what appears to be an established TCP connection already in progress, and the payload of "**../**" in every attempt. I will only provide 2 dumps for our discussion, see appendix for all 16 traces.

windump -r \giac\raw\2002.6.7 -nXvv src 66.108.153.160

14:16:47.264488 IP (tos 0x0, ttl 48, id 63896, len 301) 66.108.153.160.55804 > 46.5.180.133.80: P [bad tcp cksum d552 (->56d)!] 271305487:3271305736(249) ack 2051942796 win 5840 <nop,nop,timestamp 16830631 5820679> (DF)bad cksum 97a1 (->919b)!

0x0000	4500 012d f998 4000 3006 97a1 426c 99a0	E..-..@.0...Bl..
0x0010	2e05 b485 d9fc 0050 c2fc 290f 7a4e 298c	.......P..).zN).
0x0020	8018 16d0 d552 0000 0101 080a 0100 d0a7	.....R..........
0x0030	0058 d107 4745 5420 2f66 7930 3161 6e6e	.X..GET./fy01ann
0x0040	7561 6c72 6570 6f72 742f 2e2e 2f66 7930	ualreport/../fy0
0x0050	3161 6e6e 7561 6c72 6570 6f72 742f 6172	1annualreport/ar
0x0060	6d64 6161 2e68 746d 6c20 4854 5450 2f31	mdaa.html.HTTP/1
0x0070	2e31 0d0a 486f 7374 3a20 7777 772e 5858	.1..Host:.www.XX
0x0080	5858 2e63 6f6d 0d0a 436f 6e6e 6563 7469	XX.com..Connecti
0x0090	6f6e 3a20 4b65 6570 2d41 6c69 7665 2c20	on:.Keep-Alive,.
0x00a0	5445 0d0a 5445 3a20 7472 6169 6c65 7273	TE..TE:.trailers
0x00b0	2c20 6465 666c 6174 652c 2067 7a69 702c	,.deflate,.gzip,

```
0x00c0          2063 6f6d 7072 6573 730d 0a55 7365 722d   .compress..User-
0x00d0          4167 656e 743a 2052 5054 2d48 5454 5043   Agent:.RPT-HTTPC
0x00e0          6c69 656e 742f 302e 332d 330d 0a41 6363   lient/0.3-3..Acc
0x00f0          6570 742d 456e 636f 6469 6e67 3a20 6465   ept-Encoding:.de
0x0100          666c 6174 652c 2067 7a69 702c 2078 2d67   flate,.gzip,.x-g
0x0110          7a69 702c 2063 6f6d 7072 6573 732c 2078   zip,.compress,.x
0x0120          2d63 6f6d 7072 6573 730d 0a0d 0a          -compress....

14:16:54.704488 IP (tos 0x0, ttl 48, id 65263, len 301) 66.108.153.160.55997 >
46.5.180.133.80: P [bad tcp cksum 7db1 (->adcb)!] 290246508:3290246757(249)
ack 2065854503 win 5840 <nop,nop,timestamp 16831376 5821428> (DF)bad
cksum 924a (->8c44)!
0x0000          4500 012d feef 4000 3006 924a 426c 99a0   E..-..@.0..JBl..
0x0010          2e05 b485 dabd 0050 c41d 2d6c 7b22 7027   .......P..-l{"p'
0x0020          8018 16d0 7db1 0000 0101 080a 0100 d390   ....}...........
0x0030          0058 d3f4 4745 5420 2f66 7930 3161 6e6e   .X..GET./fy01ann
0x0040          7561 6c72 6570 6f72 742f 2e2e 2f66 7930   ualreport/../fy0
0x0050          3161 6e6e 7561 6c72 6570 6f72 742f 6172   1annualreport/ar
0x0060          6373 6f6f 2e68 746d 6c20 4854 5450 2f31   csoo.html.HTTP/1
0x0070          2e31 0d0a 486f 7374 3a20 7777 772e 5858   .1..Host:.www.XX
0x0080          5858 2e63 6f6d 0d0a 436f 6e6e 6563 7469   XX.com..Connecti
0x0090          6f6e 3a20 4b65 6570 2d41 6c69 7665 2c20   on:.Keep-Alive,.
0x00a0          5445 0d0a 5445 3a20 7472 6169 6c65 7273   TE..TE:.trailers
0x00b0          2c20 6465 666c 6174 652c 2067 7a69 702c   ,.deflate,.gzip,
0x00c0          2063 6f6d 7072 6573 730d 0a55 7365 722d   .compress..User-
0x00d0          4167 656e 743a 2052 5054 2d48 5454 5043   Agent:.RPT-HTTPC
0x00e0          6c69 656e 742f 302e 332d 330d 0a41 6363   lient/0.3-3..Acc
0x00f0          6570 742d 456e 636f 6469 6e67 3a20 6465   ept-Encoding:.de
0x0100          666c 6174 652c 2067 7a69 702c 2078 2d67   flate,.gzip,.x-g
0x0110          7a69 702c 2063 6f6d 7072 6573 732c 2078   zip,.compress,.x
0x0120          2d63 6f6d 7072 6573 730d 0a0d 0a          -compress....
```

do not pay attention to the IP and TCP checksum failures, this is do to the destination IP's being munged for study purposes

Let's try to analyze all of this now and interpret what the user is trying to do.

3. Probability the source address was spoofed: We can assume the IP address seen by the Web server is valid, and while it is easy to forge the source IP address that doesn't do the user much good if they wanted a reply back and wanted to know what was contained in the parent directory (the whole point of the "attack"). Further, we will see that it appears to be a valid TCP session taking place and we really needed one in order to initiate this request. Now, whether these are exploit attempts aimed at a vulnerable server allowing traversal or valid web requests, the user is requesting a command be issued and either wants to know what is contained in another directory or is requesting to be moved to that

directory (so we would safely assume it's not a spoofed IP since a response to the user is desired). Also, this exploit is not a buffer overflow or a DoS attack (good reasons to spoof), and it would be too much trouble to sniff replies and maintain ACKs going back and forth, especially for such a simple request.

Let's also take a look at the major TCP/IP fields to help us identify any crafting or abnormalities involved with these requests (this will also help us answer with confidence whether or not the IP was spoofed).

SYNs: It does look like the user is already in the middle of an established connection to our web server.

IP: 66.108.153.160

The source IP made 16 traversal requests to our web server. Below are the sequence numbers and at first glance you might say they appear to be part of the same connection (in other words they were not spawned from new connection attempts to the web server). What we see here is that although the next expected SEQ numbers don't match it could mean the source IP is pushing data to the server that Snort is not flagging on. But, let's look at the first sequence a little close, if we expect the next sequence number to be 3271305736 but we get 3290246508, the source IP must have pushed about 18 megs to the server in 7 seconds (doesn't seem likely does it?). But look at the next sequence, we should expect to see 3290246757 but we get 3288419197 (the number went down, not expected behavior of sequence numbers of the same connection, they should increase by the amount of data being sent). So, it appears that these numbers are part of separate connection attempts for each traversal request, or the packets are out of order.
3271305487:3271305736(249)
3290246508:3290246757(249)
3288419197:3288419445(248)
3282081897:3282082146(249)
3294305342:3294305591(249)
3304333894:3304334144(250)
3299927729:3299927979(250)
3292711234:3292711485(251)
3294188822:3294189072(250)
3298254971:3298255221(250)
3805827603:3805827834(231)
3809229612:3809229862(250)
3811573073:3811573324(251)
3819052453:3819052703(250)
3817719305:3817719557(252)
3814668903:3814669155(252)

ACKs: Notice how these ACKs do not increment sequentially in order, this is

most probably due to its TCP stack offering random SEQ numbers. If these were part of the same connection we would most likely expect the ACKs to increase according to the data sent, not increasing at one instance and decreasing another. We do see similarities between the source and destination though, when the source SYNs decrease the destination ACKs change as well from previous ACKs, again we can conclude these are separate connection requests. However, we could still say we are seeing the packets out of order or seeing previous connections requesting data after a new connection was requested. At any rate, I don't feel it is part of one connection session.

2051942796
2065854503
2065174871
2073977977
2071756078
2084042078
2079132403
2079586811
2079247510
2088034751
2601663545
2602215931
2617778459
2621168461
2622052430
2614517908

Ports: Looking at the source ports it becomes more evident that these are separate connections. Below we see the ports change in positive increments, just like we expect new connections should act. Notice though we went from port 56595 tp 41917-- why did this happen? We see from the dump above that this connection started 8 minutes later, so most likely the user closed the session and came back to it later. His system was pretty busy during that time since it looks like he rolled over the max amount of ports, or his system was able to free these high ports for this new connection. Also, you can see this coorelation with the SYNs and ACKs above too.

55997
56069
56124
56229
56390
56429
56487
56534
56595

41917
42031
42122
42258
42294
42334

TTLs: We have a TTL of 48, let's assume its initial TTL was either 60 or 64. Now, we conduct a traceroute to the source IP to try and narrow down what its initial TTL could be.

ARIN reports: inetnum: 66.108.0.0 - 66.108.255.255
OrgName: RRNY
netname: ROADRUNNER-NYC
From me this took 21 hops, so 48 + 21 = 69, this is pretty close to the predicted TTL or 64 so to me it appears this is a valid TTL value.

Flags: on the 13th byte offset of the TCP header for each dump we have a hex value of 0x18, this means that the Ack flag and Push flags are both set--- good, there is nothing unusual about this flag combination (Since the MTU is probably Ethernet and we don't send or request near the max size we don't really need to have the PUSH flag set, we do this to ensure the server processes this up to the higher level protocol fields and gives us the information (answer) we requested.) The value of 0x18 also shows us no reserved settings are set, good too.

Window Sizes: Our source IP has a window size remaining constant at 5840, and with a guessed TTL of 64 this system appears to be a JetDirect printer--- hmmm, this is unusual now.

Header lengths: Both TCP and IP header lengths checked out, the source did have TCP options (notice the 0x80 at the 12th byte offset of the tcpheader, it shows us there are 32 bytes in the header instead of the default 20 bytes, therefore we have some options,nop,nop,timestamp).

IP IDs: We can really see now that this system is pretty busy by looking at the IP IDs, which increment with every new connection (notice how many times it hit the max number of 65535 and rolled over).

63896
65263
40960
19814
51871
64325
16185
4894

36357
48049
52129
40968
25042
33788
22753
42290

Reserved fields: No reserved fields in either the IP or TCP header were found in any of the packets

IP and TCP options: We did have TCP options, utilizing the nop,nop,timestamp options. The option, length and value reported match the expected value and are legitimate options. Further, if these fields were corrupt or manipulated the destination host should reject the packet anyway. There were no IP options at all, so a clever spoof like utilizing source routing was not set.

Data: Look at what the source IP is doing in every traversal attempt, it appears the request is to get to the fy011annualreport directory, then access a different page within that directory- But, didn't the user start off in that directory anyway? Why would the user start in the fy011annualreport directory, initiate a '../' to get back to root, then change right back to that directory? Also notice the odd use of the user agent involved, it's using a RPT-HTTPClient/0.3-3 client (while not a flag by itself it is a rarely used agent, and would a printer be using this as we guess above by the TTL and window size values?). Aside from this, and the obvious us of "../" which triggered Snort in the first place, there appears to be nothing else unusual with these requests-- so what is this user tying to do, and what would be the benefit?

I also placed a Dshield database search for the source IP, I'm still waiting for the database search to be emailed to me.

4. Description of the Attack: Again Snort has alerted us on a potential attack or probe happening on the 46.5 subnet. Just like the 2 detects we have already covered, this one too appears to be a false alarm-- at least a false positive on a directory traversal attack. More than likely this is a broken bot spawned from the Road Runner search engine or it's an application testing program 'walking' through a web site and indexing all the pages accessible. As usual, without all the data to and from the systems I can only assume what is happening, especially since the user already has knowledge of the pages contained with the fy01annualreport directory. I'll try to show that the request is in essence a traversal but not an attempt to access a restricted directory and pull sensitive data. Before we begin, let me try to explain some key terms I've already mentioned: Bot, user-agents, search engines, and 'walking' a web site (or indexing).

What's a bot? A bot is a program that automatically traverses the Web's hypertext structure by retrieving a document, and recursively retrieving all documents that are referenced. Don't get these confused with Web browsers, because we operate them and they don't automatically retrieve documents. Bots are sometimes referred to as Crawlers or Spiders, although there are some differences between them all.

What's a user-agent? User-agent is the name for programs that perform networking tasks for a user, such as the agents included in Netscape Navigator and Microsoft Internet Explorer, and Email User-agent like Qualcomm Eudora, etc.

What's a search engine? Basically, a search engine is a program that uses forms to search through databases of HTML documents gathered by the bots.

What is Indexing? How does the bot know what to look for? In general the bots start from a historical list of URLs and documents with many links to other pages such as server lists and "What's New" pages. Most indexing services also allow you to submit URLs manually, which will then be queued and visited by the bot. Sometimes other sources for URLs are used, such as scanners through USENET postings or published mailing list archives. With these starting points a bot can select URLs to visit and index and then parse and use as a source for new URLs. Also, web servers can register their pages with a robot either through a URL submission forms on someone's search page or registering with <http://www.submit-it.com> who will do it for you.

If an indexing bot knows about a document it may decide to parse it and insert it into its database. How this is done depends on the robot: Some robots index the HTML titles, or the first few paragraphs, or parse the entire HTML and index all words. In our case I think the user is indexing just the HTML files for that directory, although if I were to see all the packets and analyze the amount of data we could assume the user is indexing the entire HTML page. Administrators can see if their site was visited by a bot by checking the server logs for sites that retrieve many documents, especially in a short time (in our case the user didn't retrieve too many documents, so I'm still leaning towards it was an attempt just to index the HTML documents for that particular directory). Also, the administrator can check the /robots.txt file and look for unusual User-agent header values (like the one our user used).

Let's now take a look at a common traversal attack and analyze it to see what is happening, what we should expect, and what the result would be. Then we will compare it to our dump above and determine what the user's intent was.

07/08-12:29:21.103460 attacker:1737 -> target:80
TCP TTL:64 TOS:0x10 ID:48175 DF
*****PA* Seq: 0x8CDE2D5B Ack: 0xD24163C Win: 0x7FB8
TCP Options => NOP NOP TS: 152351356 190236

```
47 45 54 20 2F 63 67 69 2D 62 69 6E 2F 66 6F 6F   GET /cgi-bin/foo
62 61 72 2E 70 6C 3F 2F 62 6F 72 69 6E 67 2F 2E   bar.pl?/boring/.
2E 2F 2E 2E 2F 2E 2E 2F 65 74 63 2F 70 61 73 73   ./../../etc/pass
77 64 20 48 54 54 50 2F 31 2E 30 0A               wd HTTP/1.0.
```

Notice the above traversal attack in this whitehat example, the user started off in the /boring subdirectory then issued 3 "../" commands effectively changing to the root directory. Then, the user changed to the /etc directory in an effort to grab the passwd file. What's happening is this web server probably has a vulnerable CGI script or server daemon that allows access to a particular portion of the file system. Because of this vulnerability the system does not properly conduct user input, and by simply adding '../' in the path the user is allowed to access either the parent directories or possibly climbing to the root directory and accessing the entire system. If the web server responds, the user is granted the passwd file which can be cracked offline at the attackers leisure-- we are dealing with a very simple yet deadly attack.

What about our data dumps above though, does it look like the user is trying to access a different portion of the file system or trying to view or execute files that could be sensitive or damaging? Take a look at them again, this user comes from the /fy01annualreport directory, issued a '../' command to go up one directory, then changes back to the original directory and issues a GET for an HTML page. The user does this 15 more times, each time retrieving a separate HTML file. Now, take a look at the user-agent involved with this request (look inside the payload), it's using an RPT-HTTPClient/0.3-3 which is not widely used. This is basically a JAVA class application emulating a browser when requesting pages (see http://www.innovation.ch/java/HTTPClient for more details); most common browsers are Microsoft Internet Explorer, Netscape, and Googlebot with Mozilla/5.0/4.0 being the most common agents involved in searches or web spider-ing. I did some searching for this agent and it appears it is linked with a Road Runner ISP search bot, probably available to their customers through a home page set up when they log on. This seems plausible, since the source IP belongs to Road Runner NYC. However, when searching on the source IP I could not definitely confirm that the IP was in fact a bot for a search engine, although quite a few people referenced it as a 'possible bot'. Here are some sites that provide a list of known or suspected bots:

http://www.sxw.org.uk/computing/robots/list.html

http://www.robotstxt.org/wc/active.html

Here is the nslookup for the source IP: 66-108-153-160.nyc.rr.com

This doesn't tell us much does it--- yes, it could be a search bot or it could be a user's cable modem. At any rate, if this is a bot maintained by Road Runner, like a lot of other automatic search engines (robots run from Google and Dogpile for example), search engines are relatively stupid and I could see it traversing out of a URL then back into the URL. Perhaps the user is searching a bunch of

documents offered up by the source, which are related and could be grouped together during multiple searches. Take a look at this URL, I searched the web for the fy01annualreport directory and it is actually linked to a technical company, along with the HTML documents the user was requesting:
<http://www.smsc.com/fy01annualreport/>
Within this site there are a bunch of hyperlinks to the HTML documents we see in our dumps. So maybe the web server on the 46.5 subnet has links or references to the SMSC site and some search bot is querying it. Either way, the HTTPClient is an API to some kind of tool, either a search engine or an automated test tool (application or vulnerability)-- this is what I think is happening---> Because the user requested every HTML page underneath the fy01annualreport directory it looks like the user is using some sort of an application testing or indexing tool which was 'walking the website' searching for all files with a particular string or extension-- we will get to some examples of this later. Also, since these pages seem to be accessible by everyone already, I don't see any logical reason to try and traverse directories in an effort to view files not designed to be accessed.

References: There are so many instances of directory traversal, affecting all sorts of web servers and CGI vulnerabilities; I've listed a few below that most closely represent the '../' usage.

CVE-1999-0842, CVE-1999-0887, CVE-2000-0436, CAN-2000-0443

5. Attack Mechanism: What can we do by traversing a directory and jumping into directories we are not suppose to see (the ones outside of the web directories)? Obviously as we have seen above a user can grab the /etc/passwd file, the same goes for the windows side by grabbing the /winnt/repair/sam file, but a user could also execute commands on the server (like the ability to install and run code, add, change or delete files or web pages, or take other actions-- reference MA-024.042001: Microsoft IIS Extended Unicode Traversal Vulnerability for more information). Path traversal attacks are normally carried out via unchecked URL input parameters, cookies and HTTP request headers. In most cases, a path traversal attack inherits the permissions of the application being executed which may then access any file allowable using those permissions. All of these attacks can be implemented simply with a user browser or user-agents emulating a browser, so in other words no need to create, write, or install special program or applications to carry out this exploit.

Here are a few Butraq listings that point to classic traversal usage and a bunch of related attacks:

BUGTRAQ numbers:
620, 689, 699, 743, 746, 772, 773, 827, 896, 921, 950, 968, 989, 1067, 1102, 1103, 1144, 1164, 1169, 1231, 1243, 1278, 1344, 1455, 1462, 1471, 1508, 1537

Below I have listed some other examples and variations of traversal attacks, and after we examine them I think we will find our data dumps do not exactly fit into exploiting a vulnerability, but rather gathering information about a site.

What about variations of traversing a directory, and what about different representations of "/" or "\" characters? What will the web server do, how will it interpret this? Notice in this example from the Savant Unicode vulnerability, we begin to see variations on the same traversal exploit. The bug within this program allows any user to change to any directory, and in most cases execute MS-DOS commands. The Savant server does filter "**/..**" requests out of the string but doesn't account for the Unicode representation of that, for example "**%2f..**".

http://www.server.com/%2f..%2f..%2f../

This is the HTTP Directory of //../../../ , in this case it is a directory listing of c:\. The only set back with this vulnerability is the user must know of a directory which allows listings, but the user could still download known files! Also, if the user knows of a directory which allows CGI-execution then MS-DOS commands can be executed:

http://www.server.com/cgi-bin/%2f..%2f..%2f../cmd.exe?+/c+dir

Here's another example of utilizing Unicode to fool the server into carrying out the request.

A request such as http://www.server.com/../../../../../../../ results in an error message on the GoAhead web server, 'Cannot open URL'. However, if the user encodes the '/' character with %2f instead it is possible to break out of the web root and read arbitrary files from the server. Here is an example:

http://www.server.com/..%2f..%2f..%2f..%2f..%2f..%2f/winnt/win.ini ---> this returns the contents of the win.ini file.

Below is essentially the same example, but we encode the "\" character instead--- some servers are vulnerable to this representation as well. In this case the Abyss web server fails to check the input, the character '\' (%5c), so the server follows the path in the URL until it reaches the file requested.

http://www.server.com/%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%2e%5cwinnt%5cwin.ini

This translates into "GET /\..\..\..\..\..\winnt\win.ini HTTP/1.0"

Below is an example of utilizing different versions of the Unicode representation of "\" or "/", look at all the variations we can have.

GET /scripts/..\xc1\x1c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc0/../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc0\xaf../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc1\x9c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%5c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%2f../winnt/system32/cmd.exe?/c+dir

And here is a sample of how the IIS server might log these attempts: Notice here

the user is trying to execute a command.

2001-05-06 12:20:19 10.10.10.10 - 10.20.20.20 80 GET /scripts/../../winnt/system32/cmd.exe /c+dir 200 -
2001-05-06 12:20:19 10.10.10.10 - 10.20.20.20 80 GET /scripts/../../winnt/system32/cmd.exe /c+dir+..\ 200 -

Whitehats has identified at least 50 variations of this attack that work against a default installation of IIS 4.0 alone!

In the appendix I have provided a couple of perl scripts designed specifically to test, exploit, and report directory traversal vulnerabilities. The first one is pretty simple and it guides you through what files you want to grab, the second is a script the does 15 separate tests on an IIS server, each test tries a different variant of Unicode representations for characters.

Okay, so we have looked at traversal attacks and many variations of the same concept from many different vulnerable systems, but do our data dumps above conform to any of these examples, and does it appear it is malicious in any way? Let's explore some other possibilities, specifically web application testing tools and indexing tools and see if they more closely represent what our user is doing. Remember what we had talked about with Indexing and bots? Indexing, 'crawling' , 'walking', or web pilfering basically all refer to the same thing, to gather as much information about web pages either to mirror the site or find key flaws and vulnerabilities in code, comments, or design. Spiders, bots, or crawlers are the automated software agents designed to carry out work that is repetitive, or because of the large scale of the request it would be impossible for an individual to undertake. They traverse the Web and navigate through links performing activities like recognizing and indexing resources or annotating broken links. And, remember the search engines use the spiders or bots to index and build databases that can later be solicited for resource discovery or retrieval.

Let's now take a look at some of these programs and see if they could be what the user is implementing on our web server. Here is a site that provides some good links and downloads for Indexing/ bot scripts, and below I've listed just a few of the applications available:
http://www.hotscripts.com/Perl/Scripts_and_Programs/Searching/Web_Indexing/

Harvest-NG: "This is a collection of Perl modules and scripts which provide a powerful web crawling and summarizing agent. The code is aimed at providing an open source, standards compliant, tool for fetching content from a wide variety of information sources, summarizing it into a set of resource descriptions, and storing these in an easily accessible database from which search services can be built and statistical information compiled."
I-Spy: "This is a Perl script which identifies new files on various remote FTP and Web sites. It grabs and compares contents of FTP directories and web pages. It will then compile a report and either send it via e-mail or save it as a web page.

I-Spy logs its activity as it chugs along. You may specify the log directory, or I-Spy will try to find one automatically. For web page reports, I-Spy will attempt to store the log in such a place where it may be referenced by the report and served by the web server."
WebAwk: "This is a proof-of-concept of a tool to automate web browsing / data collection. It works like AWK except that instead of working on files and lines it works on HTML pages and hyperlinks. It is meant to be run as a command line script and includes base_url - the URL the script was initially invoked on, base_path - root of saved data tree, url - current URL being processed, linked_from - parent of current URL, and content - the actual data corresponding to the current URL."

Here is another web crawling application, and one of the more popular ones out there:

Teleport NT/Pro: This application can mirror an entire site on your local system for futher review. The user can also search for only those files that match the criteria. For example, if the user is looking for web pages with certain key words in them such as 'email', 'contact', 'user', or 'pass' then Teleport can be set up to do this. Also, it can search for any of these extensions too like .htm, .html, .shtm, .txt, .cfm, and so on. Once a copy of the desired pages are available on the local system, the users could scour every HTML page, graphics files, and form controls to understand the design of the web site. And when the users know this it canbe extremely helpful to them if they wanted to search for an exploit or weakness.

If you want to try out your own bots to see what they can do and to understand what their purpose is, here is a great site. They provide search bots, shopping bots, tracking bots, game bots and so on: http://www.botspot.com/. If you want to learn more about indexing and harvesting web site resources, check this site out (it also provides good links to indexing and bot applications and prevention) http://www.ukoln.ac.uk/metadata/pride/wp2/d221/soa05.html

Now, if our user was implementing some version of a web application testing program or script, then this would be the site to reference: http://www.owasp.org/ Although web application testing is an up-and-coming field, owasp provides a pretty good user group and mailing list and it might not be a bad idea to run the dumps past them to get their opinion on what's going on. One of the applications they provide for web testing is called WebScarab.

"WebScarab is an enterprise level web application vulnerability scanner written in 100% Java. The tool will be able to automatically spider a web site finding potentially vulnerable web applications and then dynamically build a set of security tests for problems based on potential scenarios it finds. Types of problems will include SQL Injection, Cross Site Scripting, Cookie Poisoning and Parameter Tampering. This tool will also have an interactive proxy for manual examination as well as using the VulnXML format for 1,000's of static checks." ---> Could our user be implementing this, or some version of this, to harvest data for a later attack?

Finally, our user could be using any number of CGI or Perl scripts, or even a home-grown variation. Check out this site if you want a ton of scripts that will probe, test, index, purge, whatever a site for whatever data you want:

http://cgi.resourceindex.com/Programs_and_Scripts/Perl/Searching/Searching_Your_Web_Site/

I've tried to show you about traversal attacks and many variations of those attacks using Unicode and Metacharacters, and most likely what the attacker would be trying to do by traversing (grabbing files, manipulation, running code, and so on). I've also tried to show what else could be going on, the user could be walking a web site and indexing the HTML pages as it appears by the pages grabbed and the user-agent involved. But what about the signature of the user dumps, does his traversal request "../" and user-agent match any of the indexing or bot applications we have talked about, or the ones I searched on? Maybe, but I don't have the time to search or run every application or script on the subject, I can only interpret from the data. But, if this was an automated bot pilfering a web site we should see some evidence of this--

Above I provided 2 sites that list known bots or suspected bots and their user-agents involved in the searches (none exactly matched our users, although the user agent was used by some bots recorded from other web sites). Bots can also be identified by requesting the robots.txt file on a server (I don't see any evidence of this in our dumps, maybe the server has a log?). Bots normally index pretty fast, so looking at the server logs we should see many hits from a particular user-agent (in our case, the user walked the site and grabbed 16 HTML documents, but in an 8 minute timeframe--- doesn't seem like normal bot behavior). So what is our user using and what is our user doing? I'm still going to stick with my initial guess, that this is an ISP client using a Roadrunner search of some sort offered up through the home page when the user logs on. My second guess would be this user is using a JAVA emulation of a browser and is implementing some script to utilize exploratory web crawling to index the directory --- for what purpose, why not the entire site, why not other directories? It would be a safe bet to assume our web server is already indexed by a bot and provides links to the HTML grabbed, and our user wanted to document everything pertaining to fy01 annual reports for that company.

Was this destination IP targeted at some point? Again I looked at the past 10 days of previous detects and I could not find any recon or additional data from the source IP. Also, as mentioned before, I have noticed our web server is constantly under attack from just about every attack out there. So, maybe it's hooked up with a Google search (again, the lack of a port scan attempt or any recon) or is in fact a pointer for some bot out there since it looks like it has some references to the SMSC site. Even though this appears to be legitimate indexing traffic I would still monitor this IP and continue to analyze and suspect all traffic.

Was this a Stimulus or Response? This can be highly assumed it's a stimulus to our web server. The user is asking our web server to GET an HTML page

from a specific directory. Whether this is legitimate traffic searching a site for information to add to a search database, an exploit variant of some traversal vulnerability, or application testing, the user is soliciting the server for information and wants the answers back.

6. Correlations: I have provided some basic examples and sample traces from similar traversal attacks above, also some tools that could be used in indexing requests. Below I'm providing some links that cover pretty much everything we have discussed so far. If you typed in 'Web traversal attacks' or 'directory traversal attacks' in Google you will find plenty of correlations and user/server logs to read over. However, our data dumps do not really fit into any of these examples, and I was unable to find logs or traces of indexing bots from other users (for this particular bot).

<http://www.kb.cert.org/vuls/id/111677> -- CERT spin of IIS 4/5 directory traversal
<http://www.eeye.com/html/Products/Retina/RTHs/Web_Servers/html> -- A bunch of links to all sorts of traversal attacks
<http://www.digitaltrust.it/arachnids/IDS432/event.html> -- Unicode traversal example
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/virus/nimda.asp> -- NIMDA traversal details
<http://lists.jammed.com/loganalysis/2002/06/0060.html> -- NIMDA variant, UNICODE logs
<http://www.mycert.org.my/advisory/MA-034.092001.html> -- Server logs after successful traversal
<http://www.securiteam.com/securitynews/5VP0P2080U.html> -- Different variations of overloading UNICODE
<http://www.securiteam.com/windowsntfocus/Web_Server_Folder_Traversal_vulnerability__Patch_available__exploit_.html> -- Exploit code
<http://www.security.nnov.ru/search/document.asp?docid=3539> - Perl exploit

Dshield reports: Still waiting on the results....

7. Evidence of active targeting: I analyzed the past 10 days of detects from the source and destination IP's and did not notice any recon or any other data transfers between the two. I can assume that if the user was testing some sort of a traversal exploit or page testing they did the reconnaissance sometime earlier and waited around a while before they came back. More than likely this web server is indexed by some bot and the user conducted a search returning its web page. Also, it would be a good bet that this web page has some links to other web servers to include the SMSC site where the /fy01annualreport directory resides. The source IP is definitely targeting a specific host, but I don't think it's a malicious act. Also, I didn't see any probing or generic scans by this IP for the 45.6 subnet either. Could this be a wrong number? Not likely, especially since they have established a 2-way TCP connection, and also because the source IP

is requesting data that the user needs (for either reading, indexing, mirroring, whatever)---- in other words it's not a DoS or Trojan or a user hacking blindly.

8. Severity:

Criticality: how critical the target system is: This appears to be a DMZ web server and the router or firewall acting as the perimeter device seems to allow world-wide access to the site on this port. I will also assume the server is not serving anything sensitive or crucial so I'm going to rate this as a **2**.

Lethality: measure how severe the damage is to the target if attack succeeds: In this case, if the attacker was going to exploit a directory traversal vulnerability I would have given this a 4 or 5 depending on where the web server was installed. If the server files were installed on the same partition as the system the user could potentially jump into the SAM or passwd directories and have complete access in a matter of time. Also, the user could potentially execute programs or modify any file for that matter. If the server was located outside the system partition the severity is decreased (also if proper permissions are set on the files and directories since the user is acting under the web account) but the attacker could still have access to read or modify any file available. In our case however, the method used to search and index the web site is a normal feature, and by traversing in and out of the same directory it looks like one method of indexing the site--- and this is allowed and is okay, other than some bandwidth or CPU utilization…. But requesting only 16 pages in a matter of 8 minutes does not put too much load on either. So, I'm going to rate this a **1**.

System Countermeasures: strength measures of the target itself: I've already stated the burden of securing the site, ensuring service packs/patches are applied, auditing port/services, and logging is on the admin since our perimeter device is not filtering this service for us. I can't provide concrete evidence to how well the system is maintained, but since it does not seem to have been attacked by other traversal attempts, or malicious Unicode variants, or has responded to these attempts I'd say the server is most likely patched. However, since I've seen numerous other attack attempts from all kinds of IP's this server does offer potential access points and thus potential vulnerabilities someone can exploit latter. So, I'm going to give it just a **2** since the administrator has to keep up with this system for it to stay protected.

Network Countermeasure: Since the perimeter is not filtering this service (HTTP) or filtering what IP's can access into the DMZ, it really does not do us any good. There is an IDS attached to this subnet so if there is someone monitoring this closely we can at least have some reaction within a short amount of time. I'm going to rate this is **2** as well.

So, (2 + 1) - (2 + 2) = -1

I would recommend monitoring the source IP for future recon, indexing, or traversals to other directories, to the root directory, or jumping to directories outside what the web server should be providing. Maybe this was just a test to see if the user could walk through the site and issue basic traversal commands. At any rate, I would continue to look at the Snort alerts and compare with the server logs, and also how much data is being requested and in what amount of time (to try and narrow this down as a bot, or automated user-agent).

9. Defensive recommendation: If this were a directory traversal attack attempt, we should at least do the following to protect ourselves.

Eliminate all sample files and unneeded features from the site. Move, rename or delete any command-line utilities that could assist an attacker and set restrictive permissions on them. Be sure that the web user account does not have write access to any files on your system and permissions are set correctly throughout the partitions file structure. Use "hard coded" absolute file paths with extensions.

Good Practices: Always store data outside WWW-ROOT. Use databases instead of the file system. Never pass unchecked user input to file system commands. And do not include unchecked user input in templates or other output related files.

And most importantly for every platform or application, keep up to date with service packs, patches, hotfixes, and Bugtraq or news postings. The CVE and Butraq postings, and the reference links I listed, all provide additional information and links about prevention and defenses against this type of attack. It would also be a good idea to run a vulnerability scanner or web scripts against the web server to detect any possible weaknesses or vulnerabilities.

If this is an bot attempting to index the site there are a few things we could do if you want to prevent this from taking place. The easiest way to prevent robots from visiting our web server is to put these two lines into the /robots.txt file on the server:

```
User-agent: *
Disallow: /
```

This is saying no matter what user agent is involved with the request, do not allow it to index the site. Here are some other examples of what we can do with the robots.txt file, and how we can tailor it to what we want to allow.

```
# /robots.txt file for http://webcrawler.com/
# mail webmaster@webcrawler.com for constructive criticism

User-agent: webcrawler
Disallow:

User-agent: lycra
Disallow: /
```

```
User-agent: *
Disallow: /tmp
Disallow: /logs
```

The first paragraph specifies that the robot called 'webcrawler' can go anywhere. The second paragraph indicates that the robot called 'lycra' has all relative URLs starting with '/' disallowed. The third paragraph indicates that all other robots can not visit URLs starting with /tmp or /log.

We could also use HTML META tags to keep robots off the server instead, here is an example:

```
<META NAME="ROBOTS" CONTENT="NOINDEX">
```

If we place this within the HTML document, that document won't be indexed by the bot. Also, if we place this into the document the links cannot be parsed by the robot either:

```
<META NAME="ROBOTS" CONTENT="NOFOLLOW">
```

Here is a link for an automated monitoring application that identifies and logs web bots coming into the site:
http://www.sxw.org.uk/computing/robots/botwatch.html

10. Multiple choice test question: What log(s) show(s) an attempt to access a file outside the web directories, in an effort to traverse directories or permissions?

a. GET /cgi-bin/foobar.pl?/boring/../../../etc/passwd HTTP/1.0.
b. GET./opendirectory/../opendirectory/opendata.html.HTTP/1.1
c. http://www.server.com/cgi-bin/%2f..%2f..%2f../cmd.exe?+/c+dir
d. http://www.server.com/%5c%2e%2e%5c%2e%2e%5c%winnt%5cwin.ini

Answer: it's (a), (c), and (d)
a. shows an attempt to traverse to the root directory then change to the /etc directory (outside the web server) to grab the passwd file.
b. shows a user walking the web site, probably indexing the HTML pages offered in that directory (we see the user is already allowed to the /opendirectory and it's part of the web directories too--- so the user is not trying to access an HTML file that is restricted).
c. shows a variation of traversing using Unicode characters, the user is trying to execute a command (cmd.exe) to list directory contents.
d. shows another variation of traversing to a directory outside the web server and accessing the win.ini file, this time even the ".." are encoded as %2e.

Assignment # 3: Analyze This

Executive Summary

The University of Maryland Baltimore County through the cooperation with the SANS Institute have asked GIAC Students to analyze network logs for a period

of 5 days. These logs are generated from the Snort IDS monitor and include alert signatures, scanning logs, and out of spec alerts. I have provided analysis for all of these during the week of Oct 14 through Oct 18, 2002 to help identify external systems we should earmark for investigation and internal systems that are possibly compromised. This will help give us a baseline, or benchmark, of what may be normal traffic and what attackers are targeting, and ultimately how we should go about setting up defense recommendations. The analysis, however, cannot be considered fully complete or accurate and is based on the interpretation of the data I had to work with. I was mostly limited to observing only packet headers and alert logs based on unique university rules, so false positive or even negatives could happen since I was unable to obtain or review packet dumps for all the detects. Also, without knowledge of the university network I do not have a benchmark for normal behavior, thus I may end up tagging systems as suspicious or misuse when they might be legitimate. However, it appears there really is no security or filtering taking place because of the amount of internal systems being contacted from the outside, and IP addresses accessing the LAN that should be blocked by every perimeter router (for a list of IP networks to block visit: http://feeds.dshield.org/block.txt). Also, the extent of any security that may be implemented is just flagging on systems or events that the university considers interesting. For instance, there are tons of alerts for Watchlists that obviously are being looked at because of previous misuse or attacks, but yet the university still lets them into the network. Why are we not at least blocking these IP's at the router? Also, there are a bunch of alerts for suspicious activities that were created by the university to monitor internal systems. For example, 'TFTP external connection to internal server', 'External FTP to internal HelpDesk systems', and TFTP internal connections to external servers" are just a few alerts created to monitor if these activities occur. The question is though, why are we allowing these activities to happen? Some rules the university set up are even specific to certain internal IP's, so I can assume these activities are allowed, acknowledged they are dangerous, but allowed to be run on the network. With all of this, we need to define and decide on some security policies and acceptable use policies for the network. Further, we should implement some firewalls or at least some access control lists at the border routers. Before we continue on to the analysis, I want to provide some general comments on what I saw throughout this period. There were 1.57 Million events triggered over this 5 day period, that's just about 3.6 alerts per second every day. So we are either analyzing a severely targeted and compromised system or our IDS rules and signatures need to be looked at and tuned to better reflect the universities normal traffic-- I would bet on the latter. So again, the rules set up by the university are probably reflecting a large amount of alerts that are actually false positives, and I'll try to point these out throughout the paper and how they can take up a lot of time analyzing. There are malicious acts occurring however, and probable compromised systems on the network. I noticed several buffer overflows, worms, suspected trojans or backdoor transactions, and automated scripts and scanners hard at work. Further, while not intended to be malicious, I noticed a lot of other applications that warrant investigation such as: peer-to-peer

applications like Kazaa, Gnutella, and WinMX, FTP servers, IRC communications, IIS servers that probably are not university run, and several others I discuss throughout. Let's now move onto the numbers involved during the week.

The Numbers: Let's give a very high-level view of what we captured for this week, in terms of the amount of different events that triggered our logs.

Totals	Alerts	Scans			OOS
		UDP	TCP/SYN	Other	
Total	158954	1229756	173548	914	6702
Inside	121148	1221100	24549	3	412
Outside	37840	8656	148999	911	6308

The numbers are broken out into general types according to the data files we obtained (see below). Further, they are broken down into the amount of events either originating from the outside of the network (the Internet for example) or from inside of the network (our 130.85.0.0 address)—in fact, this becomes the footprint of my entire analysis as I'll explain in a bit. At first glance we might become very alarmed with the astonishing number of events, but let's hold off on that for just a while until we start to explain them and what they most likely mean. Much like incident handling, we don't want to 'see' something and assume it's an incident; yes these are events, because we've triggered on them, but we shouldn't cry wolf and unplug the fiber just yet--- let's investigate, and you'll see we can throw most of these events away and concentrate on the real issues like network problems, compromises, successful entries, and attacks from the inside.

List of files used to analyze:

ALERT FILES	SCAN FILES	OUT-OF-SPEC FILES
Alert021014	Scans021014	OOS_Report_2002_10_14_21815
Alert021015	Scans021015	OOS_Report_2002_10_15_13854
Alert021016	Scans021016	OOS_Report_2002_10_16_32106
Alert021017	Scans021017	OOS_Report_2002_10_17_23248
Alert021018	Scans021018	OOS_Report_2002_10_18_15331

** Note: I renamed the OOS files as 00S14.txt and so on.*

Why are there 3 separate log files per day, what do they mean, and how are each helpful in our analysis? --- I'll explain this now.

Alert Files: Simply, these are the events logged because they conform to at least one of our Snort rules. In a basic Snort install there are 40 rule files the IDS uses to analyze packets going through it. Within these files we have about 2000 rules that can trigger an event log.
Scan Files: These events are generated because the Snort.conf file (our configuration file) is set up to use a preprocessor to look for portscans, called

spp_portscan.c. Snort will use this file to go through several steps in order to classify traffic as a scan, like flag combinations or number of connection attempts within a specific period of time (see the below practical for some examples: http://www.giac.org/practical/Christof_Voemel_GCIA.txt). Obviously, if we have a lot of connection attempts in a short period of time to all kinds of ports, we probably have a port scan. Or, if we see special combinations of TCP flags this could be used for fingerprinting an OS or to try and circumvent or inject traffic through an IDS. Take a look at the below URL for the source of spp_portscan.c. http://packetstormsecurity.nl/sniffers/snort/spp_portscan-0.2.9.c

Out of Spec Files: The out of spec (OOS) file contain the packet dumps of events triggered, not just the header information we see in the Alert and Scan files. The OOS files are created when Snort finds traffic that does not meet normal standards for TCP/IP traffic. If we have occurrences of these we need to take a close look, since this could indicate malicious traffic, network problems, or applications trying to be sneaky-- either way, this traffic is anomalous in some way. For example, some of the most frequent OOS events are linked with scanning events when hosts try to identify OS's by manipulating flag settings.

Analysis: Preface to Methodology

Let's start to understand how I looked at strange traffic, marked the IP's involved, and saw if they correlated with our other sections. My process and technique for looking at the traffic (my methodology) will be explained here and carried out throughout the analysis. First off, let's start with the numbers again, it seems like management wants to see the numbers in everything we present--- It's good to be able to say, "Out of 400,000 attacks only 10% came from the inside and out of those only .01% seemed suspicious", for example. This leads to the first part in my methodology: I broke everything up into external events and internal events, then, I subdivided these groups into sources and targets for both IP's and ports and recounted everything. For instance, I first separated the logs into everything targeting MY.NET IP's (external logs) and everything coming from MY.NET (internal logs), then I cleaned up the logs and delimitated them into just source IP:port pairs and destination IP:port pairs-- explained in the 'analysis process' section. Below is an example of how I analyzed the external OOS logs (external IP's targeting MY.NET IP's).

***Please note, MY.NET all begin with 130.85 for the first two octets.**

IP to IP	**IP to destination Port**
878 209.116.70.75 130.85.100.217 389 130.85.70.183 130.85.1.4 380 200.221.192.245 130.85.91.81 199 81.86.122.65 130.85.99.174 157 204.152.189.120 130.85.168.238	3558 64.52.4.180 :21 900 209.116.70.75 :25 389 130.85.70.183 :37 380 200.221.192.245 :1214 199 81.86.122.65 :9890
IP to IP Port	**IP Port to IP**
878 209.116.70.75 130.85.100.217 :25 389 130.85.70.183 130.85.1.4 :37 380 200.221.192.245 130.85.91.81 :1214 199 81.86.122.65 130.85.99.174 :9890 157 204.152.189.120 130.85.168.238 :113	11 200.221.192.245 :4374 130.85.91.81 10 200.221.192.245 :4867 130.85.91.81 10 200.221.192.245 :4818 130.85.91.81 10 200.221.192.245 :4723 130.85.91.81 10 200.221.192.245 :4499 130.85.91.81

IP Port to IP Port	**IP Port to Port**
11 200.221.192.245 :4374 130.85.91.81 :1214 10 200.221.192.245 :4867 130.85.91.81 :1214 10 200.221.192.245 :4818 130.85.91.81 :1214 10 200.221.192.245 :4723 130.85.91.81 :1214 10 200.221.192.245 :4499 130.85.91.81 :1214	11 200.221.192.245 :4374 :1214 10 200.221.192.245 :4867 :1214 10 200.221.192.245 :4818 :1214 10 200.221.192.245 :4723 :1214 10 200.221.192.245 :4499 :1214
IP Port Source	**IP Port Destination**
11 200.221.192.245 :4374 10 200.221.192.245 :4867 10 200.221.192.245 :4818 10 200.221.192.245 :4723 10 200.221.192.245 :4499 10 200.221.192.245 :4466	878 130.85.100.217 :25 389 130.85.1.4 :37 380 130.85.91.81 :1214 331 130.85.6.40 :25 199 130.85.99.174 :9890 157 130.85.168.238 :113
Port # Source	**Port # Destination**
13 :4818 13 :4374 12 :4723 12 :4233 12 :4170	3558 :21 1240 :25 401 :1214 389 :37 199 :9890
IP # Source	**IP # Destination**
3558 64.52.4.180 900 209.116.70.75 389 130.85.70.183 380 200.221.192.245 199 81.86.122.65	878 130.85.100.217 389 130.85.1.4 380 130.85.91.81 334 130.85.6.40 200 130.85.99.174

This is how I analyzed every separate log file, in all I had 10 separate data sheets to look at; 5 external and 5 internal sheets covering Alerts, TCP Scans, UDP Scans, Other Scans, and OOS logs. Basically, for each type of event-- alerts, scans (TCP, UDP, Other), and OOS-- I analyzed them both externally and internally. I used this approach to try and hone in on attacks from the outside, compromises from the inside, and normal traffic. There was just too much data for me to analyze as a whole and I didn't want to miss something hiding inside all that data. Plus, this method has some good advantages as well:

IP to IP: This obviously show us the amount (1) source went to (1) destination

IP to Port: The intent of this is to narrow down scans, possible backdoors to a compromised destination system, or normal traffic-- what ports is the IP targeting?

IP to IP Port: This really starts to distinguish other traffic from scans-- if it's a port scan to a single IP it wouldn't show up in this as a high number since this is asking, "what source IP is targeting a destination IP:port pair? Also, if it's an IP scan it won't show up here either.

IP Port to IP: The intent here is to find automated scanners, packet crafting, backdoor applications, or even DoS or overflow attacks--it also can show who is doing a lot of transfers. If one IP has the same source port targeting a destination IP we should look at this closer since we would expect the source

ports to change with each new connection. But, it also could show an already established connection (a socket) doing a lot of transfers. Finally, it could show returned traffic to the destinations, example would be the destination system surfing the web where the data would look something like this: 64.110.103.132:80 --> 130.85.24.50.

IP Port to Port: Now we really start to distinguish perhaps normal traffic from anything malicious. This can show an automated scanner or backdoor application more clearly while normal web traffic should start to disappear from this list. We still would see established sessions pushing a lot or data like file sharing or steaming applications and even DoS or overflow attacks-- but we should see this more clearly too.

IP Port to IP Port: This should help us solidify all of our data above and clearly weed out general port or IP scans and normal traffic.

IP:Port Source: This will help me to see if any IP's are targeting anyone else from the same source port, that would be odd. I should expect the same general number from the data above, but if it's greater I know he went somewhere else.

IP:Port Destination: This helps me to find any compromised destination systems. I would compare this number to who is going to it, if only one IP is targeting this then I look at what he's doing, if a lot of IP's are going to this then something is odd, or it's a server port perhaps.

Port # Source: This should reiterate the above numbers, if there is a spike in a certain port then I look into this to see who else is using it.

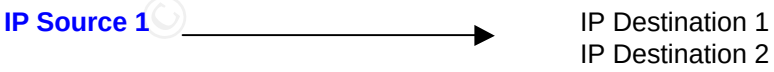
Port # Destination: This helps me to see if other source IP's targeted other destination ports, or shows us a port scan targeting a particular service.

IP # Source: This should show us the same numbers, but if a new IP shows up then we should know he's going to a lot of separate IP's and Ports-- like a scan.

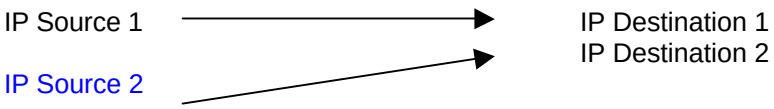
IP # Destination: This also should show us the same numbers, but if a new IP is listed here then we know it was the target of a lot of separate source IP's, like a scan or maybe it's a server.

I would then analyze the numbers to see what's going on using the following method: Who is the source IP going to? Who else are going to those IP's? Who else are those IP's targeting? Who else is going to those targeted IP's? Then finally, Who are those IP's targeting? The diagrams below explain it better, and this becomes the basis for me to find correlations and compromises:

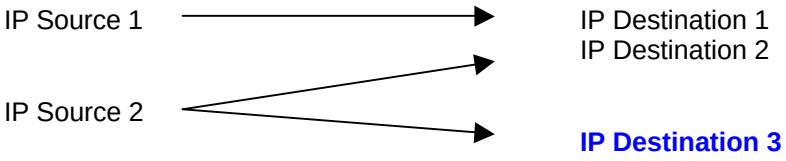
Who is the source IP going to?



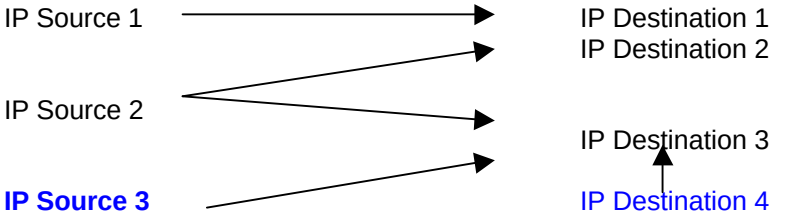
Who else are going to those IP's



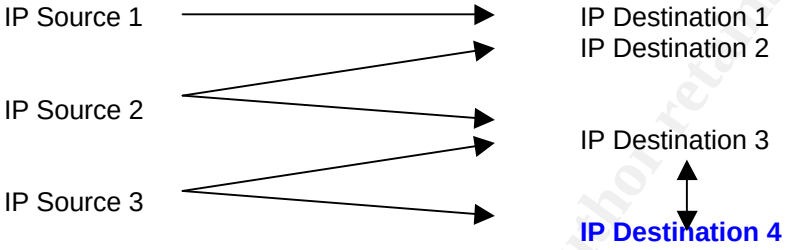
Who else are those IP's targeting?



Who else are going to those targeted IP's?



Then finally, Who are those IP's targeting?



Since I found Source 3 went to Destination 4 then that's odd, especially since Destination 4 is communicating with Destination 3. And if I find Source 3 is using a backdoor or something malicious then we must analyze Destination 3 closely, and further analyze what Source 2 is really doing, and so on. This may seem confusing but for me it helps to really see what's going on and hopefully get a handle on correlations. So, based on this redundant and somewhat overkill approach I come up with the IP's I want to investigate, the IP's I feel may be normal traffic, and the IP's that may have networking issues--then I compare to the Snort alert logs. Also, I try to concentrate on internal systems participating either in attacks or accessing the outside in unusual ways. Let's now dive into the detail and take a look at the alerts first and the IP's I want to investigate and why.

Alerts External

Signature (click for sig info)	# Alerts	# Sources	# Dests
Watchlist 000220 IL-ISDNNET-990517	89321	70	66
SMB Name Wildcard	17151	487	901
spp_http_decode: IIS Unicode attack detected	3596	172	497
Incomplete Packet Fragments Discarded	3537	21	18

FTP DoS ftpd globbing	1741	13	2
IDS552/web-iis_IIS ISAPI Overflow ida nosize [arachNIDS]	1396	1309	541
Queso fingerprint	1203	95	26
Watchlist 000222 NET-NCFC	677	36	40
High port 65535 udp - possible Red Worm - traffic	532	85	52
SUNRPC highport access!	523	34	41
Null scan!	316	37	19
EXPLOIT x86 NOOP	209	26	27
Tiny Fragments - Possible Hostile Activity	173	7	7
High port 65535 tcp - possible Red Worm - traffic	130	5	5
CS WEBSERVER - external web traffic	123	29	1
SMB C access	96	50	17
Port 55850 tcp - Possible myserver activity - ref. 010313-1	90	17	15
NMAP TCP ping!	68	18	22
EXPLOIT x86 stealth noop	54	4	4
EXPLOIT x86 setuid 0	32	24	18
EXPLOIT x86 setgid 0	25	21	16
External RPC call	23	1	23
spp_http_decode: CGI Null Byte attack detected	23	3	1
Possible trojan server activity	21	5	5
Bugbear@MM virus in SMTP	14	12	2
Port 55850 udp - Possible myserver activity - ref. 010313-1	11	7	7
RFB - Possible WinVNC - 010708-1	11	3	6
Attempted Sun RPC high port access	8	4	6
TFTP - External TCP connection to internal tftp server	6	3	4
TFTP - External UDP connection to internal tftp server	6	5	5
External FTP to HelpDesk 130.85.70.50	6	6	1
External FTP to HelpDesk 130.85.70.49	5	4	1
EXPLOIT NTPDX buffer overflow	4	4	4

TFTP - Internal UDP connection to external tftp server	4	4	3
SYN-FIN scan!	4	1	1
TFTP - Internal TCP connection to external tftp server	3	1	1
External FTP to HelpDesk 130.85.83.197	3	3	1
Probable NMAP fingerprint attempt	2	1	1
CS WEBSERVER - external ftp traffic	1	1	1
Back Orifice	1	1	1
Fragmentation Overflow Attack	1	1	1

Top Talkers

Using the methodology above these are the IP's I decided to concentrate on, I'll also provide diagrams to help explain why I choose them. *** Top Talkers will include systems with; interesting detects, misuse of resources, network issues, compromises, or any combination of these.***

69563 212.179.103.7 130.85.70.91 Watchlist 000220 IL-ISDNNET-990517

This IP registered almost 70,000 alerts, nearly 57% of all the alert traffic. Notice the alert type is labeled Watchlist, this is the only information we are given. Watchlist means that any packet from either from this IP or the subnet class coming into the network will be flagged as this general type of alert (in our case any IP coming from the 212.179 network will be flagged). In other words, this may not be malicious traffic at all, but was flagged in the past at some point for some reason, maybe this subnet was misbehaving before (then why not block them?) or maybe they are the source of a lot of network traffic (which seems more likely in our case). Let's find out who this IP is:
nslookup: cablep-179-103-7.cablep.bezeqint.net [212.179.103.7]

RIPE.net:

```
inetnum:        212.179.100.0 - 212.179.124.255
netname:        CABLES-CONNECTION
mnt-by:         INET-MGR
descr:          CABLES-CUSTOMERS-CONNECTION
country:        IL
```

Well, it's a cable modem ISP hosted from Israel, good reason to be on a Watchlist. Let's try to figure out what's going on with this now. Out of his 69,591 alerts 69,563 went to one host, 130.85.70.91. Out of this 69,494 were from only 3 separate source ports and went to 3 separate destination ports.
30194 212.179.103.7 :1162 130.85.70.91 :3029
28041 212.179.103.7 :1551 130.85.70.91 :2685
11259 212.179.103.7 :1153 130.85.70.91 :2325

This is odd, I could see a lot of alerts from multiple source ports to one destination port or vice versa (a busy connection for example) but why did the connection jump ports? Let's now look at the time involved to see if we can get a better view (maybe the connections are from different times or days which would explain the different ports, but it doesn't explain who the server is---does the application us a random port to listen on, or does it have a portmapper maybe, or are there 3 separate services?). On Oct 16 he did his first connection at 17:51 and it lasted only a minute. Then he came back at 04:33 on Oct 17 and stayed until 12:23 that day. The first time he used port 1162 was at 04:33 on Oct 17. The first time he used port 1551 was at 07:34 on Oct 17. And he used port 1153 at 06:31 on Oct 17. Hmmm, so what happened on Oct 16? He initially started the whole thing with a connection from port 1553 to port 2184 (maybe this is return traffic we are seeing since the watchlist probably only triggers for inbound connections from this subnet). So what is all of this? Maybe he's using cleo client/server apps, or an SNA server (both of which use port 1551) or maybe it's ftp disguising itself using PORT commands like 212.179.103.7,6,15 for example (if you don't get this check out how to break out an FTP trace). I really don't know what the answer is but it does, kind of, look like the original connection attempt didn't work since it only lasted a minute. Then the server (probably the source) changed its server port to 1162 and ultimately to 1153 until they were happily transferring (although it looks like they were already doing pretty good with port 1162???). Of course it could have been an overflow of some kind then the attacker was messing around with backdoor ports, again I'm not sure about this. What does IANA say about the ports?

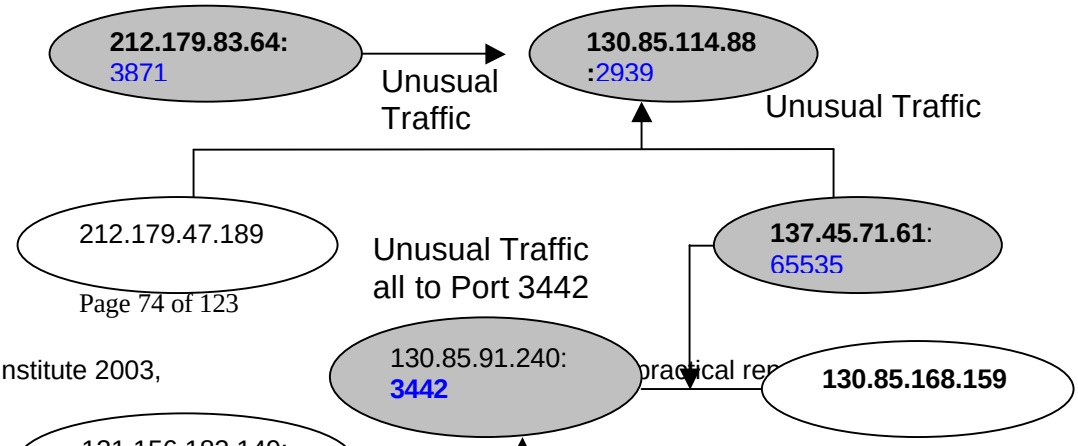
1162	health-trap	1551	hecmtl-db
1153	Unassigned	3029	LiebDevMgmt_A
2685	mpnjsocl	2325	Dsgn Sp. Lic. Mgt

So even maybe they are doing some administration or device management, but highly unlikely. At any rate, we should investigate this IP for some compromise.

I grouped the following IP's together since they all caught my eye once I started to analyze the second most frequent source. Below is the diagram.

12360 212.179.83.64 130.85.114.88 Watchlist 000220 IL-ISDNNET-990517

17	137.45.71.61
128	130.85.91.240
34	130.85.88.165



212.179.83.64 is another Watchlist log registering 12,360 alerts. So let's try to figure out what he's doing as well. Out of all the alerts every one went to one IP, 130.85.114.88, and on the same source port (3871) to destination port (2939)--seems to be a busy connection. The connection began at 02:43 on 14 Oct and lasted until 05:11 that day; he never came back or initiated any other connections to MY.NET. Again, I'm not sure what's going on here without some packet dumps and I'm not sure what these ports are used for (we see unusual ephemeral socket ports like the first IP we analyzed). IANA reports port 2939 as SM-PAS (I don't know what this service is). Could this be a UDP scan trying to elicit router ICMP messages to discover the network, ACL's and protocol and ports open? Maybe, if the router governing this subnet is a perimeter router or a server farm router, otherwise why only scan 1 system and where are all the ICMP messages (is the router silenced or Snort not set up to capture this)? Let's now look at who else went to 130.85.114.88. We see that 2 other IP's connected to this MY.NET system, 212.179.47.189 and 137.45.71.61, and both connected to the same destination port of 2939. Who is 137.45.71.61?

```
OrgName:    Radford University
OrgID:      RADFOR

NetRange:   137.45.0.0 - 137.45.255.255
CIDR:       137.45.0.0/16
NetName:    RU-NET
NetHandle:  NET-137-45-0-0-1
Parent:     NET-137-0-0-0-0
NetType:    Direct Assignment
NameServer: RUSERVE.RUNET.EDU
NameServer: RUACAD.AC.RUNET.EDU
NameServer: RUCS.RUNET.EDU
```

Also, this IP came in from source port 65535-- this is suspicious in itself. If we drill down we see this IP also went to 2 other MY.NET systems, 130.85.168.159 and 130.85.91.240, both on the same source port of 65535 (now we know this is odd since a new connection should statistically use a different port). I noticed that this IP went to port 3442 on 130.85.91.240, again this is suspicious traffic to a very unusual port. IANA reports this port as the OC Connect Client, whatever that is, but at any rate the OC Connect server listens on port 3477 which we don't see. If we drill down even farther we see that 130.85.91.240 was connected to by 4 other systems, 131.156.182.149, 68.83.182.149, 212.179.104.60, and 212.179.104.61, all to port 3442. Also, can you notice any similarities between these IP's? Notice the first 2 have the same last 2 octets (odd but not impossible) and the last 2 are only off by 1 number. The first IP belongs to a Northern Illinois University and the second IP is a Comcast address. If I were to take a stab at figuring out what this is all about I might guess it's misfire and someone is spoofing this IP and we are really seeing the return SYN-ACKs (that could explain the odd ephemeral ports we are seeing as a socket as the attacker is picking random high number ports hoping they won't be flagged, as opposed to picking common server ports-- this could also explain our first analysis). However, we see 131.156.182.149 used source port 65535 to connect, and that will be flagged as Red Worm traffic. If we drill down even farther with the IP's that connected to 130.85.91.240, we see 212.179.104.61 went to 130.85.88.165 to port 4509 (again we have odd ephemeral sockets). Also, 2 other IP's connected to this, 158.227.65.57 from port 65535 and 212.179.98.93. The first IP (a Spain origin) tried to connect to port 6257 (maybe WinMX media sharing) and the last IP tried to connect to 1214 (probably Kazaa). My conclusion with all of this is to investigate 130.85.114.88 because of the unusual port connects (if it's misfire then we're okay), 130.85.91.240 for the same reason (and because we have 2 systems connecting using source port 65535), and 130.85.88.165 for all the reasons above and since it appears it's hosting multiple media sharing applications.

2533 202.102.233.93 130.85.112.204 Incomplete Packet Fragments Discarded

This one is interesting. The alert is telling us that the IP datagrams are fragmented but incomplete (meaning the packets were never assembled entirely or Snort never saw the initial fragmented packet). It would appear they are using Snort 1.9 with the Frag 2 preprocessor to make Snort a stateful monitor. So, who is this IP?

```
inetnum:      202.102.224.0 - 202.102.255.255
netname:      CHINANET-HA
descr:        CHINANET Henan province network
descr:        Data Communication Division
descr:        China Telecom
country:      CN
```

There are many DoS exploits that could be carried out using this fragmentation, and that theory could be justified by the amount of traffic between the 2 IP's. I

also noticed this IP went to 2 other MY.NET systems, 130.85.163.235 and 130.85.109.64. Let's look at the time involved to see if we can shed some light on this. I first saw this IP connect to 130.85.163.235 on Oct 14 at 16:01 and ended the connection less than a minute later. Then on Oct 15 he came back to this IP for the last time at 13:31 and stayed until about 16:55. On Oct 16 at 14:20 he first started connecting to 130.85.112.204 and stayed until 19:18 that day. What's interesting during this time is a single alert at 18:20 from source port 65535 to destination port 33790 (flagged as Red Worm trafic)..hmmm. Then on Oct 18 at 13.45 he first went to 130.85.109.64 and stayed for under a minute as well--interesting--and never came back to it. Also on Oct 18 he went back to 130.85.112.204 and stayed until 17:45 that day. Again, the very interesting thing during this time was a single Red Worm alert at 15:51, but this time the source port was 42718 to destination port 65535--this is clearly odd. Now I wanted to know who else was connecting to 130.85.112.204. I found that only 2 other systems went to it, 61.144.56.215 and 202.102.249.118 (another China IP from the same subnet--odd). Well, let's see what these 2 IP's are doing. The first IP is flagged with the Incomplete Packet alert we see above, from and to port 0, and made no other connections. The second IP came from source port 38527 to destination port 65535-- Red Worm again (I can most likely say we have a strong correlation now that the destination MY.NET IP is compromised). I think what really makes this a compelling argument is the Red Worm traffic again appeared out of nowhere, as a single alert, only about 14 seconds after the communication from 61.144.56.215. The only thing I don't understand is why 65535 is both a source and destination to 130.85.112.204. If it's really Red Worm traffic I'm not seeing the regular pattern, or it's not being logged. Adore (or Red Worm) is a worm that scans for hosts from random Class B subnets, trying to exploit vulnerable systems running either BIND, wu-ftpd, rpc.statd and lpd services. If a vulnerable host is found, it attempts to download the main worm part from a web server in China (hmmm, that's where 202.102.233.93 is from). Then the newly infected system will run the *pscan* program against another randomly generated subnet and the associated ports (*pscan-bind* on port 53, *pscan-statdx* on port 111, and *pscan-lprng* on port 515). It also sets up a backdoor port on 65535 that awakens if an ICMP packet with a payload of 77 Bytes is sent to it. Finally, it will email the attacker pertinent information such as configuration files and passwords. However, I don't see any of this kind of traffic coming from the MY.NET system, or again it's not being logged from the inside. Further, I don't see any ICMP packets sent to either, but really that's futile since the attacker could have used any IP (spoofed) to send the ICMP to awaken the backdoor port if ICMP is allowed into the network (I'm not sure it is because we really don't see any ICMP alerts, and we should at least see some). Also, I don't see any initial reconnaissance to it looking for vulnerabilities on ports 53,111,21, or 515 (maybe the attacker used another compromised system that might not be flagged to do the scanning, a slow scan just to these ports, or did his reconnaissance at an earlier time-- this would be more likely since he's mostly targeting this system). And the last thing I checked were any outbound connections to port 25 (sending an email) from the MY.NET system, but that turned up empty. What else could

this be? At first I thought this could be the product of a poor performing router since we have 2 systems with incomplete packets (perhaps the router governing the source IP or the MY.NET subnet is bad, and since I don't see what the importance is of this system I don't see why this would be a DoS to it). But then I thought, if it were a bad connection why would the source continue to try for hours, any other user would give up. Also, I didn't see any ICMP or UDP large packets or anything similar to this attack or anything reported by a router to the source IP's (maybe it's silenced?). So, maybe it ***is*** a Dos of some kind, certainly 202.102.233.93 is sending enough packets at a steady pace to justify this, but 61.144.56.215 doesn't send enough to cause a DoS. But, if it were a Dos then why did he try to connect to the system (Red Worm) in the middle of this--that doesn't make sense? FireWall-1 has a vulnerability that any UDP packets traveling through VPN-1 destined to any host port 0 will reboot, but we don't see this because the attacker continues to connect for hours. Now, it could be an automated tool for scanning, but then why did the China IP only go to 3 IP's? Further, he came back on multiple days and hours, if he was scanning he should have got all the information needed in a matter of minutes (and it's not a slow scan because of the amount of alerts and occurrences). We know that hping2 with the "-r" option can set the source port to 0 and even Nmap can use this with protocol scans (but again with this theory why don't we see any ICMP messages and why does the scan connect for hours and days?). Could it be a buffer overflow attack against the MY.NET system? Maybe, but then again why did it take so long, and if it set up a backdoor port why would he initialize the buffer overflow again the next day-- that just does not make sense. I would have to conclude that this is probably a bad router other than the very odd single Red Worm packets injected in-between sessions. I would certainly inspect this system for compromise based solely on that.

1957 130.85.83.94

This is a MY.NET system as the destination for 1957 alerts, most coming from another Watchlist IP, 212.179.8.78 (in fact, all of its traffic goes to the MY.NET system). As usual, we see odd ephemeral socket connections from this subnet, this time it appears the MY.NET system is hosting something on port 2394. IANA reports this port as ms-olap2, or the MSSQLServerOLAPService (msmdsrv.exe uses this port for the SQL Server OLAP service). I decided to check out the source ports used for this connection to try and see if this is really the case (this could be return traffic that was initiated from inside). Here are the top 10 source ports used, all going to port 2394:

356 :3187 99 :3703
258 :3234 93 :4087
136 :3847 90 :3628
128 :3795 88 :3829
112 :3874 75 :3821

Obviously, this doesn't look like the connections originated from the inside and thus we are seeing returned traffic, it's very doubtful the source IP is hosting

these services and it doesn't look like a scan from the internal system since this is an odd port range to scan (3100's to 4000's). When I looked at the times I saw the source IP first used port 3187 on Oct 16 at 10:45 until 11:01:45, then at 11:01:51 he used 3234. He continued changing source ports, all to higher sequential ports than the previous, until the connections stopped at 13:37 that day. This looks like normal behavior for a system making new connections or accessing different parts of an application that spawns a new socket. Could this be a DoS against the internal system, yes it could be although I'm not aware of a specific vulnerability for this service/port even though it's part of SQL that has many problems like unprivileged use, DoS, unchecked buffers, and poor security checks. Perhaps it's a basic SYN Flood against this service, but I checked the scan logs and didn't find any alerts for this IP. However, I did notice the MY.NET system had scan alerts and it too was using odd ephemeral sockets, also it used UDP port 2394 as a source in scans to numerous outside IP's, all to different classes and subnets (almost 400 alerts were generated) and to all sorts of destination ports (mostly to Kazaa port1214). So, I'm going to assume the MY.NET system is indeed hosting something on port 2394, but that it's probably compromised or is running some sort of a file-share server application that searches for shares. I also took a look at who else is going to this IP and I saw 212.179.86.141 went to port 80 and 67.36.84.5 went from port 80 to destination port 2394. To sum this up, we should take a look at this system for compromise.

1751 130.85.100.158

This is another MY.NET system generating a lot of alerts. Let's dive into who's connecting to it and to what service. By far, the most connects came from 80.137.155.155 generating 1218 alerts. RIPE reports this IP belongs to:

```
inetnum:        80.128.0.0 - 80.146.159.255
netname:        DTAG-DIAL16
descr:          Deutsche Telekom AG
country:        DE
admin-c:        DTIP-RIPE
tech-c:         ST5359-RIPE
status:         ASSIGNED PA
```

Out of these alerts all were from the same source port, 2104, to the same destination port, 21. Okay, this appears to be a pretty busy FTP session, but let's try to see if that's really happening. First I'll take a look at the times involved. The first connection came in on Oct 14 at 01:50 (odd time to do FTP transfers) and lasted until 05:11 that morning (this is a long time for normal FTP sessions, unless we are transferring a lot of data or we are connected over a slow link). Apart from this I can't really give much more detail on what's happening, the connection is steady and on the surface appears normal (I wish I could see the packet dumps). So, let's look at what Snort is reporting. We see this traffic was flagged as an 'FTP DoS ftpd globbing' attack--this is quite possible. Let's now take a look at what this rule is looking for, and what ftpd globbing means. Globbing is the process of expanding wildcard notations into complete file

names. For example, if we issue a *.txt we will be returned with all the files ending in .txt--simple enough. Globbing can also include certain characters into system-specific paths, and can be used by various shells too. There are many exploits that use vulnerable globbing functions to execute a DoS, overflow, or gain root privileges. Check out http://www.cert.org/advisories/CA-2001-07.html for numerous platforms affected. For instance, Microsoft FTP contains a vulnerability in the STAT command with a pattern-matching (glob) code. This vulnerability could be exploited to execute a Denial of Service attack. The DoS attack will bring down the FTP service and it must be manually restarted to restore operation. In our case this doesn't appear to be the case since the user continues the connections for hours. If this were the attack, we should be looking for the following in the payload "STAT ?*<240 x X>". Another example involves a buffer overflow vulnerability based on the way a wu-ftp server handles globbing requests. The wu-ftp service contains a heap corruption vulnerability that may allow an attacker to execute arbitrary code if a specially crafted glob request is sent, even if the user logged in anonymously. Check out http://www.ciac.org/ciac/bulletins/m-023.shtml for a detailed analysis and exploit code. Yet another example could be a simple request such as “LIST */../*../*/../*/../*” that could cause the server to crash. I don't think the Snort rule triggering this alert is set up for any specific globbing exploit, it's probably written to flag on any wildcard used in the payload. So, based on this information I am thinking this is either a normal FTP session or an overflow attack to gain root privileges. It could be a regular DoS too if averaging 6 alerts per minute is enough to prevent normal connections, and I don't really think it's a SYN flood since that's not enough attempts to hang a system (plus I don't see any evidence in the scan logs of this either). And since the source IP didn't connect to any other IP (at least not flagged) I can assume he's not trying to search and exploit other systems (malicious acts or testing an exploit or gaining a zombie). So, if the source is attacking he's pretty interested just in this 1 MY.NET system, and because I don't see any other evidence of him in any other logs I can assume he's done the reconnaissance sometime earlier, or this FTP server is already known by him or advertised on the web. Next, I took at look at the top 5 IP's also going to this FTP server, and all went to destination port 21.

```
320    80.11.105.159 130.85.100.158
103    217.81.230.56 130.85.100.158
 34    213.39.130.170 130.85.100.158
 25    193.251.78.186 130.85.100.158
 14    80.141.127.182 130.85.100.158
```

80.11.105.159 started his connection on Oct 15 at 04:44 and continued pretty constant until 05:16 that morning. He started off on source port 1992 for a few minutes then ended his connection. Then, 7 minutes started another connection this time on port 2461--this shows me the source system is pretty busy since his system ran through 469 ports (remember, it's normal to increase the port number for every new connection or socket). But, this doesn't look hostile so I took at look at IP 217.81.230.56. This IP started off on Oct 17 at 20:06 and stayed until

21:05 that day, his connection was regular but not a constant interaction. He used and stayed on source port 1921 throughout the duration. The rest of the IP's all stayed for at least a couple of minutes but no longer than 15 minutes and again, nothing looked really abnormal about the sessions (source ports changed consistent with time gaps in the connection--like a new connection-- and the time of day was pretty consistent with work hours). Next, I took a look at what the FTP server was doing and what else was targeted to it. I noticed that the MY.NET system was included in some of the internal UDP scan alerts.

3	130.85.100.158 :1082	208.185.54.23 :53 -also doing mail exploits
2	130.85.100.158 :53	65.119.25.162 :64886
2	130.85.100.158 :53	65.119.25.162 :36930
2	130.85.100.158 :53	64.15.251.198 :14418
2	130.85.100.158 :53	64.14.117.10 :30488
2	130.85.100.158 :53	64.14.117.10 :15992
2	130.85.100.158 :53	64.0.96.12 :61776
2	130.85.100.158 :53	213.61.6.2 :58660
2	130.85.100.158 :53	213.61.6.2 :33335
2	130.85.100.158 :53	212.62.17.145 :56721

At first I thought the FTP server was doing DNS strike-backs to verify the IP addresses trying to connect to it, but then I realized it's coming from port 53 NOT to a DNS server, and none of the IP's connecting to port 21 are listed-- hmmm, is this system also an internal DNS server? I doubt it otherwise wouldn't we see a lot more alerts to it or from it, and who would put an FTP server on with a DNS server? So, my next thought was maybe these were responses to the destinations performing DNS scans, but that didn't seem likely since the destination IP's really didn't have any other alerts and didn't really go to any other IP's (Except for 208.185.54.23 :53, he was also performing SMTP exploits). I'm not sure what this is about what we need to check this IP for what is bound to port 53 (LSOF or Fport can help us). Let's now see what else was targeted to the FTP server.

- 3 instances of *IDS552/web-iis_IIS ISAPI Overflow ida nosize*
- 8 instances of *spp_http_decode: IIS Unicode attack detected*

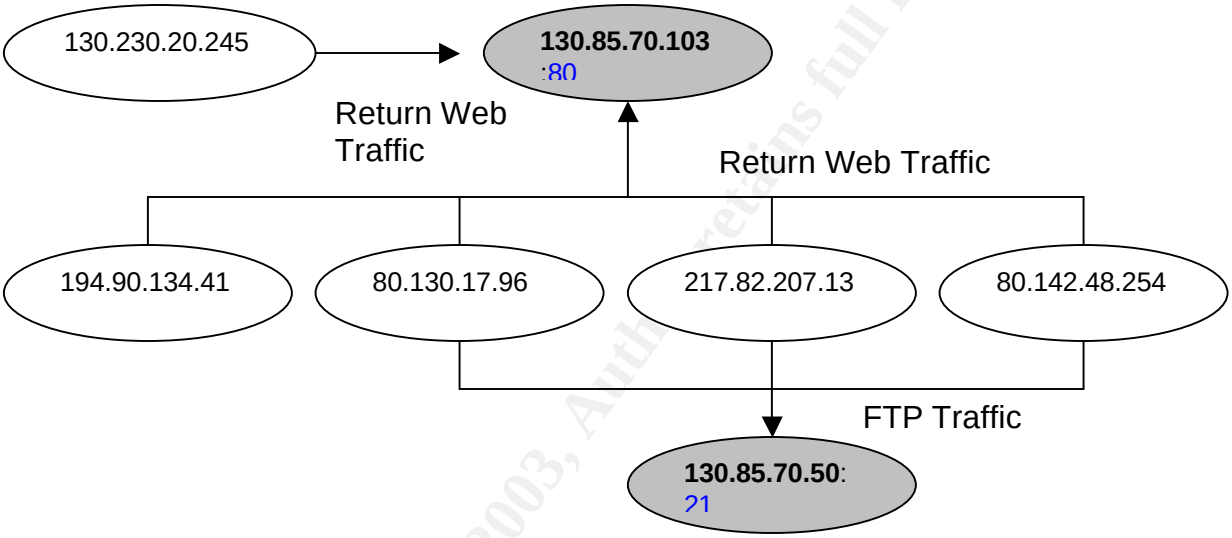
And here are the IP's involved in this:

4 211.97.104.57 130.85.100.158 :80
2 208.164.8.12 130.85.100.158 :80
2 68.0.81.88 130.85.100.158 :80
1 219.162.138.211 130.85.100.158 :80
1 212.2.223.165 130.85.100.158 :80
1 202.99.184.173 130.85.100.158 :80
1 151.200.88.30 130.85.100.158 :80

This leads me to believe the FTP server is actually an IIS server running the FTP service, I'm not going to go into detail about the above alerts but conclude that we should ensure the services that are suppose to be on this system and that they are patched and secured, especially if IIS is involved.

1357	130.85.70.103
6	130.85.70.50

These are grouped together also because of the odd connection associated between the two. Here is a diagram to show the relationship.



Let's introduce the main source in this connection, 130.230.20.245.

```
OrgName:    Tampere University of Technology
OrgID:      TUT

NetRange:   130.230.0.0 - 130.230.255.255
CIDR:       130.230.0.0/16
NetName:    TAMNET
NetHandle:  NET-130-230-0-0-1
Parent:     NET-130-0-0-0-0
NetType:    Direct Assignment
NameServer: RESSU.CC.TUT.FI
NameServer: NS-SECONDARY.FUNET.FI
```

This IP from Finland had 1144 alerts, all targeting the MY.NET system 130.85.70.103, and all flagged as *spp_http_decode: IIS Unicode attack detected*. I looked at this alert for a while but didn't notice the source doing anything really suspicious even though we have a bunch of alerts. This alert is generated by Snort's http_decode packet pre-processor, and states it is used to decode HTTP URI strings and convert their data to non-obfuscated ASCII strings-- take a look at Detect 3 in Assignment 2 above for analysis on Unicode vulnerabilities and exploits. But basically, it can allow attackers to read documents outside of the web root, and potentially execute commands via malformed URL's containing

Unicode. But like I stated in assignment 2 these can be false positives if the user is using Netscape or FrontPage viewers or certain user agents to walk through web pages. This alert is also suppose to help us identify Nimda and Code Red alerts, check out CVE-2000-884, http://www.incidents.org/react/code_redII.php, and http://www.incidents.org/react/nimda.pdf) for analysis on worm infections of Microsoft's IIS web servers. Once infected, the host will scan port 80 looking for other web servers to infect. I checked all the alert and scan logs for the MY.NET system but it didn't register anything else, so I don't think it's infected with the worm and it doesn't appear it has been compromised (no unusual port connections either). The problem may be just a high number of false positives, this shows us the problems in using signature-based rules not specific enough to narrow down what we want to capture. But I decided to take a look at who else is connecting to this system anyways. From the diagram, all of the other 4 IP's going to the MY.NET web server seem to be conducting normal web transfers on the surface. I then decided to look at their source ports just to make sure they were not using an automated scanner or crafting packets, and they seemed normal too, increasing sequentially every time they accessed a different page on the server. So, I conducted my final check, "Who else are they going to?" just to make sure, and I noticed that 3 out of the 4 IP's also went to 130.85.70.50 to port 21, and one went to 70.49 on port 21. This triggered the simple alert "External FTP to HelpDesk 130.85.70.50", a rule the university set up obviously. What's unusual then about all of this is why the helpdesk system is allowed to host an FTP site in the first place, what on Earth could this be used for? Anyway, what's even more unusual is the fact the 3 IP's all went to the web server one to two seconds right after they went to the FTP server. Perhaps the FTP server has a link to the web page for some reason, but even so you'd expect a little more of a delay between the alerts for the user to actually implement the link-- but since it happened so quick something is odd about this, and suggests automation. But why are these systems related somehow? These 2 systems need to be checked for how they interact with each other, and for any compromises.

356	212.179.27.6
664	130.85.168.192
101	130.85.88.243

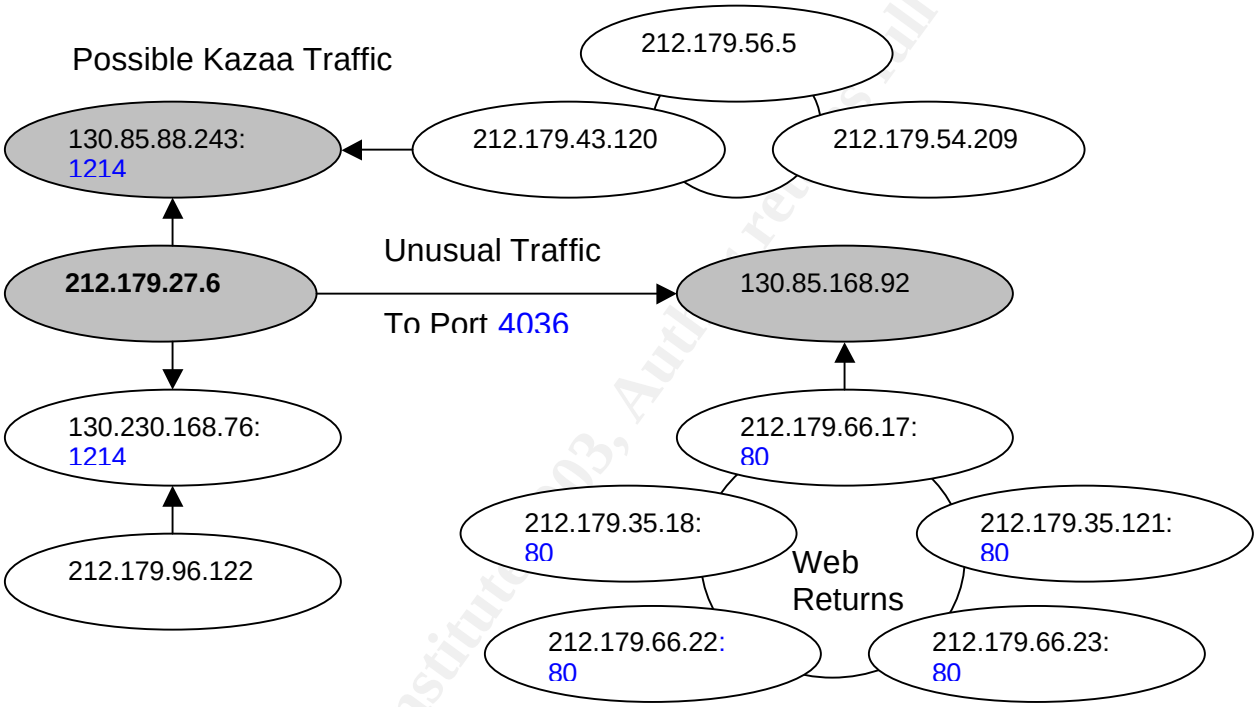
These are grouped together because of the relationship between the source. Again we have a Watchlist IP connecting to the MY.NET system 130.85.168.92, all to port 4036. We see this IP first connected on Oct 17 at 14:31 and had a pretty continuous session until 14:55. He switched source ports a few times during this duration but it seemed consistent with new connections based on the time gaps between the port before it and how it increased. Here is a list of the ports involved during the session:

224 212.179.27.6 :1836 130.85.168.192 :4036
52 212.179.27.6 :1905 130.85.168.192 :4036
36 212.179.27.6 :1945 130.85.168.192 :4036
14 212.179.27.6 :1937 130.85.168.192 :4036

5 212.179.27.6 :1927 130.85.168.192 :4036

Could this be return traffic we are seeing, and the MY.NET system really initiated a scan of some sort? This seems unlikely as this is an odd port range to scan for-- so what is this? I've heard that this port can be used for an ASP module for Apache server administration but I can't confirm that. IANA reports port 4036 as 'WAP Push OTA-HTTP secure'. What's this? This could be a proxy gateway sending data it received from a wireless phone using the over-the-air (OTA) protocol. The wireless application protocol (WAP) is what the phone uses for it's data. The connection-oriented *push* requires that some point-to-point (the client contacting the gateway) connectivity be established before the push content can be delivered (what the phone is transmitting), either using the WSP or the HTTP service. The gateway is able to request an origin to initiate connectivity to by sending a certain content type using a connectionless push. There are 2 ways it can do this, an un-secure or secure push, either using port 4035 or in our case 4036. So, could the MY.NET system be an origin from a gateway, or even its own gateway-- I guess that's possible but very odd. There were several other Watchlist systems generating alerts to this IP so I decided to analyze the top 5 for what they were doing (remember, the only alert I was given is that they are on a Watchlist, so they may or may not be doing anything malicious at all). After reviewing their traces and times it appears all the alerts are actually return web traffic back to the MY.NET system (in other words, it was surfing the 212.179 web sites). On the surface it would appear we have 5 systems coming from port 80 and attacking a bunch of ephemeral ports on the internal system, but it just reiterates my theory that the sensor only triggers on incoming alerts from these IP's (we need to fine-tune this). So, let's now look at what else the original source is doing. From the diagram below we can see it went to 2 other MY.NET systems, both to destination port 1214 (Kazaa file-sharing application). Well, where do we start to explain the vulnerabilities and concerns using this application, we could do an entire paper just on why we shouldn't be running this. For instance, there is a DoS vulnerability where a remote user can cause the software to consume 100% of the CPU resources by sending large messages in 4096 byte chunks. In fact, the attacker doesn't even need to be logged into the Kazaa network to make this happen, just aim it at the port of service by issuing telnet or netcat for example. I've provided a proof-of-concept script I found at http://www.securitytracker.com/alerts/2002/Jul/1004843.html in the appendix. Also, there is a security vulnerability that causes KaZaA to launch its ads in the local zone (the trusted zone your browser uses). This could allow an attacker to execute scripts embedded inside the ads to elevate privileges, do damage, or retrieve sensitive data--- that's not good. And yet another example is a DoS against the fasttrack person-to-person service used by Kazaa that can make the system hang. Also, identity spoofing and theft can be used against this service. What happens is the Kazaa's client opens a pop up window every time a message is received and the username is accepted, however the memory is not checked against available memory so it's possible the system will exhaust its resources with too many window pop-ups. Kazaa uses HTTP for sending the requests between users using tags embedded in the http header and the GET

command (I talked about this is assignment 2, detect 2). I provided a sample trace in the appendix of how Kazaa sends requests and the explanation of this, along with how you can spoof the identity-- thanks to Josh (josh@pulltheplug.com), omega (mtwoar@hotmail.com) on July 25th, 2002. Well, who else are going to these Kazaa clients? We can see from the diagram that 4 other Watchlist systems are also connecting to the Kazaa ports. So, maybe the relationship between the original source IP to 130.85.168.192 is not an ASP module or an OTA transmission, but some kind of file-sharing application or even a gaming port? At any rate, we need to check our what this system is really running and monitor it for possible compromise (start with FPort again). Also, be sure to check the policy on hosting file-sharing applications like Kazaa running on 130.85.88.243.



380 130.85.80.144

This is another MY.NET system that was the destination for the Incomplete Packet Fragments Discarded Alert we already talked about. Two main IP's generated these alerts, 140.142.8.72 and 140.142.8.73--that's odd. Again we see the destination and source ports are both 0. Below I listed the other alerts for the internal system as the destination.

3 instances of *IDS552/web-iis_IIS ISAPI Overflow ida nosize*

5 instances of *spp_http_decode: IIS Unicode attack detected*

There were 2 IP's that generated most of these alerts, 195.55.126.10 and 211.90.236.230-- they were also doing a lot of other IIS exploits against

numerous MY.NET subnets, they are obviously scanning and we need to keep an eye out for them. I also checked the internal alerts and scan logs for the MY.NET IP to see if it was infected or sending out crafted packets and I didn't find any evidence of this, so it seems it's just the recipient for all of this but we should check it out just like the other IP's getting these alerts.

There are a few other top IP's by numbers that I didn't talk about, and I'll list them here. It should be noted that although I don't see anything really malicious (other than scanning) we should mark these IP's for future monitoring.

465	61.176.37.16	This IP is doing SMB NetBIOS Scans
372	68.0.81.88	This IP is doing web scans against the MY.NET network
1479	212.179.35.118	This IP to the best of my analysis is simply returning web page requests to 8 separate MY.NET systems, but I would still monitor this IP
409	212.179.112.101	This IP to seems to be just returning web requests
702	212.179.66.17	This too appears to be web returns

Other Alerts Of Interest

Below I will list other alerts I have not mentioned yet, along with a brief explanation and the top IP's involved. These are less numerous alerts or are alerts with less correlation (like scans against a particular service). These should be considered as malicious and important in the overall investigation of internal systems.

IDS552/web-iis_ IIS ISAPI Overflow ida nosize
This is a well-known exploit against IIS default installations where it installs the index server for categorization of files. However, the index server has an unchecked buffer in the ida.dll file that handles the URL requests. Since the Index Server runs with the same permissions as the System account an attacker who exploits this will have full control of the system. For a complete detailed analysis of this exploit check out: (http://www.eeye.com/html/Research/Advisories/AD20010618.html).

Source	# Alerts (sig)	Destinations	# Alerts (sig)
62.154.234.2	13	130.85.70.207	13
203.94.243.1	5	130.85.21.54	8
62.181.183.196	4	130.85.21.44	8
213.244.90.26	3	130.85.150.83	7
211.75.2.236	3	130.85.180.36	7

Queso fingerprint
The Queso fingerprint scan is accomplished by sending a series of seven TCP packets to the target host. Six of these packets contain TCP flags and options that do not conform to standard TCP rules or have inconsistent combinations meant to elicit some response from the target. Since a TCP stack differs with each operating system, they will respond differently to theses odd packets. The responses received from the targeted host are compared to the scanners files of know signature for how a particular stack will respond. Snort flags this alert by checking for really only one of these odd combinations, the presence of the reserved bits being turned on (R0 and R1). I've provided some traces of this alert I found in the OOS files, and you can see what it's flagging on-- but at the same time we can see how this general rule causes a lot of false alarms.

Source	# Alerts (sig)	Destinations	# Alerts (sig)
209.116.70.75	316	130.85.100.217	281
204.152.189.120	63	130.85.24.21	267
212.43.244.24	57	130.85.24.23	260
193.170.194.22	50	130.85.6.40	152
65.214.43.159	45	130.85.168.238	63

10/13-02:36:19.313587 209.116.70.75:34629 -> 130.85.100.217:25
TCP TTL:50 TOS:0x0 ID:21374 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xBC30BA0 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 929650910 0 NOP WS: 0

10/13-07:10:45.067276 66.216.110.142:44047 -> 130.85.6.40:25
TCP TTL:50 TOS:0x0 ID:64148 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x2EB3E551 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 260186053 0 NOP WS: 0

10/17-17:38:05.050487 204.152.189.120:42360 -> 130.85.168.238:113
TCP TTL:52 TOS:0x0 ID:26885 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x6659E618 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 110399261 0 NOP WS: 0

I took a look a quite a few of these dumps for the 3 IP's I've provided traces for. I noticed everything seems normal like the sequence numbers change, IP ID number changes, TTL's are realistic and so on-- so these are not connection retries and don't appear to be crafted other than the use of the reserve bits with the SYN flag---Notice we have a value 12****S*. Here we see why the Snort rule flagged this as Queso, but the use of ECN reserved bits is not a sole determination of scanning or crafting anymore as routers become aware of how

to notify one another of congestion on the network. A lot of people stated noticing that this rule was producing a bunch of false positives based on legitimate ECN usage by Linux systems. Whitehats posted an updated rule for Queso that changed the TTL of the rule to look for values greater than 225 since Queso sets it's initial TTL to 255. So, let's take a look at the traces above again and we see we have TTL values of 50 and 52, so the initial TTL was probably 64 which is the TTL Linux usually uses. So, maybe this is a false positive-- but then I looked at the Window size of each (0x16D0 in hex is 5840-- that's not a default window size for Linux). So, although I would still assume this is a false positive based on normal options, TCP behavior, and the time the alerts were logged, I still cannot fully say this isn't Queso. Below I've provided a typical Queso trace from Whitehat.

12/22-13:40:07.346966 source:16720 -> target:80
TCP TTL:255 TOS:0x10 ID:48057
S*****21 Seq: 0x61FFCC46 Ack: 0x0 Win: 0x1234

And here is the updated Snort signature.
alert TCP $EXTERNAL any -> $INTERNAL any
(msg:"IDS29/scan_probe-Queso Fingerprint attempt"; ttl:>225; flags: S12;)

Null scan!
This is classified under the 'stealth' port scans because this utilizes TCP packets with no flags set. It's intent is to try and circumvent some firewalls or routers that block TCP based on their flag settings. When the packets reaches the target they elicit RST packets for ports that are closed and no reply for ports that are open (since they silently discard these). Just like the Queso scan, there is no defined way to respond to these type of packets so each OS will generate a unique signature when responding. So, the null scan is not only used to map ports, but to try and guess remote OS's. Below is a trace that was flagged as a Null Scan alert.

10/15-04:45:26.426068 218.43.208.59:1396 -> 130.85.91.240:40195
TCP TTL:110 TOS:0x0 ID:32863 IpLen:20 DgmLen:40 DF
******** Seq: 0xA40E4500 Ack: 0x5FA1622 Win: 0x3C38 TcpLen: 0

The only odd thing about this trace is the sequence number is not zero like we would expect in a traditional null scan, and it looks like we are responding to a session since the Ack has a value too? This might be a false alarm based on some gaming and file-sharing apps that seems to manipulate packets in this manner. For instance, I took a look at 130.85.185.48 and analyzed it like I have the other IP's and came to the conclusion it's actually Gnutella traffic manipulating the TCP packets. In addition to null scan alerts it received other alerts all related to some manipulation of TCP packets:

10/14-04:15:16.798750 [**] Queso fingerprint [**] 68.45.255.126:3040 -> 130.85.185.48:6346
10/14-05:10:44.702594 [**] EXPLOIT x86 setgid 0 [**] 161.53.248.94:2864 -> 130.85.185.48:6346

10/14-05:40:17.864353 [**] EXPLOIT x86 setgid 0 [**] 193.204.89.1:3581 -> 130.85.185.48:6346
10/14-05:48:53.113705 [**] NMAP TCP ping! [**] 64.119.138.2:80 -> 130.85.185.48:6346

Here are some references for this type of scan:
http://www.nwconnection.com/2001_03/pdf31/cybercrm31.pdf
http://www.synnergy.net/downloads/papers/portscan.txt

Source	# Alerts (sig)	Destinations	# Alerts (sig)
64.231.170.76	212	130.85.168.239	212
195.46.65.63	13	130.85.185.48	25
198.76.77.172	9	130.85.6.40	16
68.83.182.149	6	130.85.84.147	14
218.226.246.253	5	130.85.91.240	10

SUNRPC highport access!

This alert represents an attempt to access the high ports that RPC normally handles and maps to the appropriate service requested. However, an external user could just try to scan for these services instead of trying to connect to the RPC portmapper port to solicit information. Examples of such services are TCP 32773 (rpc.ttdbserverd), 32776 (rpc.spray), 32777 (rpc.walld) and 32779 (rpc.cmsd). Many exploits and vulnerabilities exist for RPC services like cmsd, statd, and ttdbserverd, check out the links below for general security threats using RPC.
http://www.sans.org/newlook/resources/IDFAQ/trouble_RPCs.htm
http://www.securitybugware.org/mUNIXes/4537.html
http://www.cert.org/advisories/CA-2000-17.html

Source	# Alerts (sig)	Destinations	# Alerts (sig)
68.55.246.114	118	130.85.149.14	118
170.140.204.81	62	130.85.139.46	62
66.187.232.100	50	130.85.83.61	51
66.187.232.56	42	130.85.82.31	39
193.147.152.102	37	130.85.163.25	37

EXPLOIT x86 setuid
EXPLOIT x86 setgid
EXPLOIT x86 NOOP
EXPLOIT x86 Stealth NOOP
All of these alerts fall under the category of buffer overflows affecting the x86 or Intel architecture. A buffer overflow occurs when a program or process attempts to write more data in a buffer, a temporary memory storage, than it is programmed to hold. The excess data can then be pushed into adjacent and separate buffers thus corrupting (or crashing) that data or overwriting the valid data in buffers (code injection).
The EXPLOIT x86 setuid is an attempt to exploit a Unix system by setting the userid to 0, or root access. The signature for this alert is to look for the content "b017 cd80" in the payload of the packet. EXPLOIT x86 setgid is essentially the same thing but the payload is "b0b5 cd80". I think the most common use of this exploit is against the wu-FTP service. Here are the Snort rules:

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 setgid 0"; content: "|b0b5 cd80|"; reference:arachnids,284; classtype:system-call-detect; sid:649; rev:5;)

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 setuid 0"; content: "|b017 cd80|"; reference:arachnids,436; classtype:system-call-detect; sid:650; rev:5;)

EXPLOIT x86 NOOP will be flagged if 0x90 NOP sled was found in a packet. The 0x90 NOP sled is used in buffer overflows to kind of compensate for where a stack will end once it has been exploited. In other words, the attacker will send his attack code with a NOP sled. Then once the service is exploited it will try to return data from the stack but will be presented with a bunch of 0x90's and do nothing until it can reach the beginning of the **attacker's** code. EXPLOIT x86 Stealth NOOP is essentially the same thing but implementing the NOP with different values (eb 02). Below are the Snort rules for both of these attacks.

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 NOOP"; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth: 128; reference:arachnids,181; classtype:shellcode-detect; sid:648; rev:5;)

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 stealth NOOP"; content: "|eb 02 eb 02 eb 02|"; reference:arachnids,291; classtype:shellcode-detect; sid:651; rev:5;)

Here is a summary of how many alerts were associated with each type of exploit, and how many sources and destinations were involved.

Signature	# Alerts	# Sources	# Dests
EXPLOIT x86 NOOP	209	26	27
EXPLOIT x86 stealth noop	54	4	4
EXPLOIT x86 setuid 0	32	24	18
EXPLOIT x86 setgid 0	25	21	16

Tiny Fragments - Possible Hostile Activity

If we see Tiny fragment alerts then it's either going to be malicious activity, network problems, or the need to pass packets through a network segment that is smaller than the packet trying to be transmitted. However, the last reason might not be as valid anymore since most networks ride on Ethernet that has a MTU of 1500 Bytes, so if we see really small packets then there is a problem. Some of the malicious things small fragments can do is circumvent an IDS or filtering device by having packets so small that information like port numbers are contained on subsequent packets. Also, they can be used to create fragment overlaps that overwrite the port number of the first fragment with the desired port the attacker wants, again in an effort to bypass an IDS. They also can be used to crash TCP stacks that cannot understand crafted overlapping fragment offsets. And finally, they can be used to implement a DoS by making the target use its memory to reassemble a ton of tiny fragments-- but we should probably see ICMP Reassembly time exceed messages if this is occurring. Neohapsis posted a guideline for a logical threshold (size of fragments) for triggering this alert: http://archives.neohapsis.com/archives/snort/2000-05/0103.html
Also, Snort has a preprocessor called "defrag" that can be set up to trigger on these:
frag2: IP defragmentation support

This preprocessor performs IP defragmentation. This plugin will also detect
people launching fragmentation attacks (usually DoS) against hosts. No
arguments loads the default configuration of the preprocessor, which is a
60 second timeout and a 4MB fragment buffer.
Reference:
CVE-1999-0683, CVE-1999-0804
http://www.snort.org/docs/faq.html#1.5

Source	# Alerts (sig)	Destinations	# Alerts (sig)
68.55.87.49	91	130.85.168.80	91
68.83.182.149	68	130.85.91.240	68
65.95.81.173	8	130.85.182.135	8
218.6.37.6	3	130.85.100.10	3

68.42.122.189	1	130.85.88.155	1

Port 55850 tcp/udp - Possible myserver activity
Throughout the paper I identify systems either using this port or going to it, so I'll just briefly describe this alert. Myserver is a distributed denial of service tool that will install a rootkit with trojan versions of ls and ps, DDoS tools, and set up a backdoor shell on port 55850. However, false positive are likely since the university set up this rule to only alert us whenever an instance of this port is seen.

Signature	# Alerts	# Sources	# Dests
Port 55850 tcp - Possible myserver activity - ref. 010313-1	90	17	15

Possible trojan server activity
This is another Snort rule the university set up to at least look for port 27374, the subseven trojan. I talk about this alert later on in the paper and the IP's involved.

Bugbear@MM virus in SMTP
For a complete and detailed analysis of his exploit visit:
http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html

Generally, the W32.Bugbear@mm is a mass-mailing worm that spreads through network shares in MS environments and attempts to terminate antivirus and personal firewall processes. Additional 'features' it has is the capability to perform keystroke logging and set up backdoor shells.

CVE References: CVE-2001-0154

Source	# Alerts (sig)	Destinations	# Alerts (sig)
212.135.6.14	2	130.85.6.40	13
212.40.5.186	2	130.85.145.9	1

RFB - Possible WinVNC - 010708

WinVNC (Virtual Network Computing) is a remote desktop access application developed by AT&T which allows a user to view the targets desktop. The current version shipped is vulnerable to a buffer overflow attack where a specially crafted response has the ability to obtain access to a client's machine and execute arbitrary commands as the user running the client software. Please reference http://www.uk.research.att.com/vnc/ for further details on this exploit.

Source	# Alerts (sig)	Destinations	# Alerts (sig)

24.189.104.14	9	130.85.83.54	3
66.200.114.148	1	130.85.104.209	3
68.33.45.145	1	130.85.84.160	2

spp_http_decode: IIS Unicode attack detected
spp_http_decode: CGI Null Byte attack detected
The first alert I covered extensively in assignment 2, detect 3, and also above when I identified the IP from Finland, so I'll just talk about the CGI exploit. This alert indicates that within the content of the packet, a string of %00 was found to try and elicit some information to be displayed that otherwise would not run-- basically it's trying to confuse a script about where the end of a URL's input is. Rain.forrest.puppy talked about this exploit in Phrack, http://www.wiretrip.net/rfp. So, utilizing this type of Unicode attack we can obtain commands, files, and traverse directories that we shouldn't be accessing. From the Snort FAQ's and some tech posting it's common for cookies and SSL traffic to produce many false positives. This alert can be turned off by utilizing the cginull switch in the http_decode preprocessor.

Source	# Alerts (sig)	Destinations	# Alerts (sig)
68.55.192.221	15	130.85.29.3	22
216.0.177.35	6		
152.163.188.71	1		

IP's as MY.NET as the source

Source	# Alerts (sig)	Destinations	# Alerts (sig)
130.85.87.67	2095	216.241.219.14	2095
130.85.153.145	811	209.10.239.135	1112
130.85.86.102	206	143.48.220.84	210
130.85.163.119	168	205.188.137.80	125
130.85.153.174	167	207.189.75.40	96

Exploit NTPDX buffer overflow
This alert deals with the network time protocol (port 123) and a possible buffer overflow of the daemon. is vulnerable to a buffer overflow exploit. Since the NTP service runs as a root process, exploiting this can lead to complete control of the system-- and since it's a buffer overflow the initial attack can be from a spoofed IP. Please reference the following links for more information:
http://www.linuxsecurity.com/advisories/netbsd_advisory-1255.html

http://online.securityfocus.com/archive/1/174011

Source	# Alerts (sig)	Destinations	# Alerts (sig)
195.92.252.254	1	130.85.84.100	1
63.250.219.152	1	130.85.88.164	1
64.7.192.181	1	130.85.117.25	1
216.148.215.98	1	130.85.111.11	1

Alert: NIMDA - Attempt to execute cmd from campus host
This is an attempt to execute cmd.exe to a targeted server, so the presence of this in the payload will trigger this alert. The target IP belongs to MS and is a search page: http://www.microsoft.com/downloads/search.aspx?displaylang=en?
This is probably a false alarm since so few alerts were generated by the IP's involved.

Source	# Alerts (sig)	Destinations	# Alerts (sig)
130.85.86.19	1	65.54.250.120	2
130.85.157.52	1		

Alerts Internal

Signature (click for sig info)	# Alerts	# Sources	# Dests
spp_http_decode: IIS Unicode attack detected	32346	368	683
spp_http_decode: CGI Null Byte attack detected	4299	59	73
High port 65535 udp - possible Red Worm - traffic	676	18	53
IRC evil - running XDCC	257	1	6
Port 55850 tcp - Possible myserver activity - ref. 010313-1	63	12	13
Port 55850 udp - Possible myserver activity - ref. 010313-1	41	6	3
Possible trojan server activity	22	6	6
High port 65535 tcp - possible Red Worm - traffic	21	4	6
TFTP - Internal UDP connection to external tftp server	13	4	4
RFB - Possible WinVNC - 010708-1	7	5	2

Incomplete Packet Fragments Discarded	6	1	1
HelpDesk 130.85.70.50 to External FTP	3	1	1
TFTP - Internal TCP connection to external tftp server	3	1	1
TFTP - External TCP connection to internal tftp server	3	3	2
NIMDA - Attempt to execute cmd from campus host	2	2	1
HelpDesk 130.85.83.197 to External FTP	2	1	2
connect to 515 from inside	2	1	1
Bugbear@MM virus in SMTP	2	2	2
HelpDesk 130.85.70.49 to External FTP	1	1	1

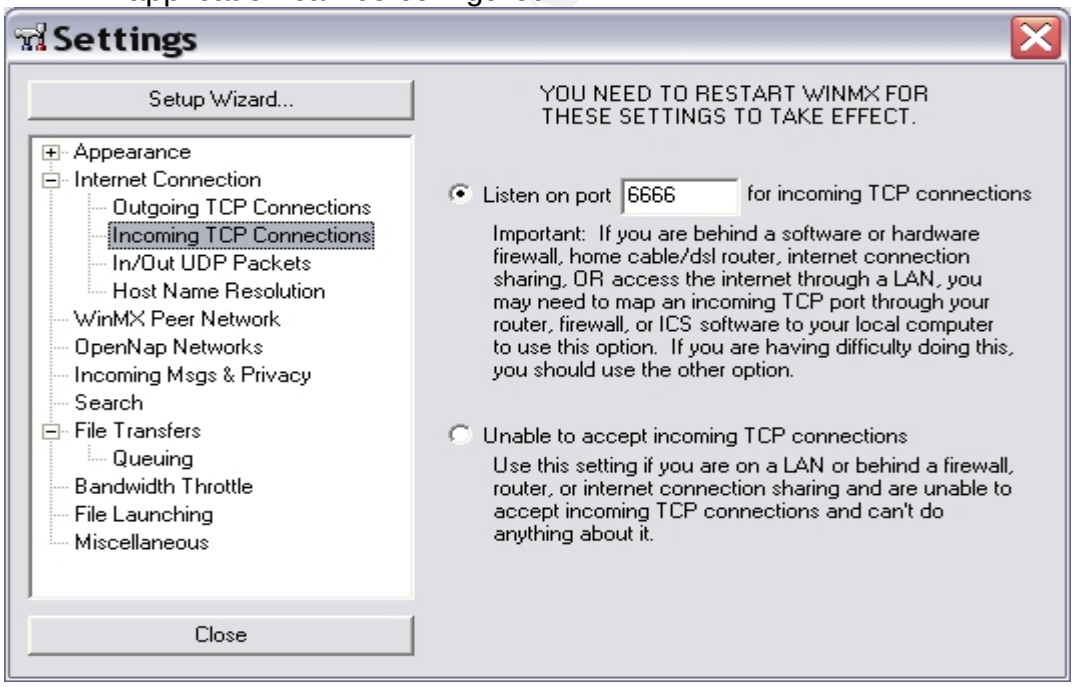
Top Talkers

I have already identified some internal systems above that we should investigate for possible compromises, network issues, or policy issues. The analysis will now try to solidify additional MY.NET systems we should be concerned about, and provide correlation between the IP's I've already identified in the external alert section.

325	130.85.83.146
120	130.85.70.176
118	130.85.165.24

I grouped these together because of the relationship between them and the same port pair combinations. Let's talk about the first IP, 130.85.83.146. Out of the 325 alerts generated by this IP, all were from source port 6257 targeting destination port 65535. This is clearly odd, especially since he went to 6 separate destinations (this is not normal port behavior). Our Snort rule has triggered this as **'High port 65535 UDP - possible Red Worm traffic'.** Here are the destination IP's the MY.NET system went to:
80.117.99.19
219.102.101.68
65.67.244.134
68.63.19.188
142.59.163.1
12.213.233.51
We've already talked about port 65535 in the external alerts section, so I decided to search these IP's for any correlation with external alerts. First, I found that each IP also had external alerts to the MY.NET system, so this tells me they are communicating back-and-forth (but, I still can't tell who is initiating the traffic?).

Second, I noticed that the IP's also went to several other MY.NET systems using the same port combination (this time the ports are reversed obviously, 65535 to port 6257). Among the several MY.NET systems as destinations, 130.85.70.176 was a target common to all of them. So, I next took a look at what else 130.85.70.176 was doing and all of the alerts had the same port combinations, and most went to 158.227.65.57--> and this IP did the same thing to other MY.NET systems, it's like a vicious circle! So, because I saw so many of the same port combinations I decided to search just on this and weed out any other IP's I might be missing, and I stumbled across 130.85.165.24. The reason I had not noticed it before is because he's not targeting any specific IP, but is using this port combination to go to many external IP's (this looked like a scan of some sort). So what's going on with all of this, who is really initiating this traffic-- is this a worm, game port, file-sharing application, or what? I really don't know the answer without looking at some packet dumps, unfortunately the OOS files didn't contain any captures for this type of traffic so I have to resort to what IANA and Snort says the ports are. Remember our discussion above about port 65535, Snort reports this as Red Worm traffic but again I don't see this conforming to the Red Worm signature (again, the MY.NET systems are not scanning random class B networks for vulnerabilities and I don't see any ICMP traffic). Some games use this port so maybe that's what it is, it could be anything really but it's all suspicious. Port 6257 is a common port used by the WinMX file-sharing application. Could this be what this is all about? Let's take a look at how the WinMX application can be configured.



Notice under the Internal Connection menu the user can configure what TCP port to listen on and which one to connect to, same goes for UDP packets (It even gives us so tips on how to map through routers and firewalls). By default WinMX uses UDP port 6257 for transfers-- so how could port 65535 come into play?

Well, obviously the user can configure this but take a look at this snip from the WinMX support page--
"The port numbers chosen...can be substituted for other numbers if needed...Pretty much any number can be used instead of these, but try to avoid using numbers less than 5001, as these are generally used by the operating system and other common services. The range of available ports that should be used are between 5001 - 65535."
So, could we be seeing users picking 65535 because that's what the tech page is telling them to do--most users I think would pick one of these 2 numbers just because they are listed. Or maybe this is indeed some variant of the Red Worm or another worm searching for file-sharing applications. At any rate, we should investigate all 3 of these MY.NET systems for compromise. Additional evidence that helps us identify these IP's as systems we want to investigate is all 3 had a bunch of scanning logs as well. 130.85.83.146 scanned a ton of times to port 6257, 122,429 times, and to a bunch of other ports all on source port 6257. He didn't target anyone in particular; in fact he went to 22,221 separate destinations. 130.85.70.176 had 78674 scanning alerts, most from and to port 6257, and to 21126 separate destinations. He also targeted port 6699 (that's odd and does not look good). The same pattern goes for 130.85.165.24, and he went to 26733 separate destinations-- and he too targeted port 6699.
Below I've provided some common ports used with similar file-sharing applications:
Aimster: TCP 5025
Bearshare: TCP 6346
Direct Connect: TCP 411/412
eDonkey: TCP 4661/4662; UDP 4665
Gnutella: TCP 6346
Grokster: TCP 1214
Hotline: TCP 1234/5498/5499/5500/5501
KaZaA: TCP 1214
LimeWire: TCP 6346, 6347
Morpheus: TCP 1214
Napster: TCP 6699
ToadNode: TCP 6346
WinMX: UDP 6257, TCP 6699
Xolox: TCP 6346

259 130.85.100.220

This system is very interesting, what really caught my eye wasn't the fact he's contacting IRC ports but how he contacted it. First we will look at the destination IP's this system is contacting all to port 6667.
216.12.211.209
64.45.60.200
62.13.43.58

66.250.105.173
12.233.14.31
Fortunately, these IP's are not involved in any other alerts, either internal or external. So let's start analyzing the connection to the first IP. The MY.NET system contacted this IP 127 times, 76 times from source port 2613 and 51 times from port 2783. I decided to take a look at the times involved in order to figure out why we have a port shift (maybe they are 2 separate connections in time). He first connected to the external system on Oct 18 15:48 and continued off and on regularly until 20:21 that night. What I noticed were the time gaps between the sessions, every time he used source port 2613 he was starting a new session based on the time between the alert before it. Then, he would almost immediately switch to port 2783 (either that or 2 processes start at the same time and bind to the same IP and destination port). He then started back up at 20:27 and stayed until 23:53--this is very odd, what suspicious times to start and IRC channel. Also, right towards the end of the second session he TFTP'd to 12.233.14.31 at 23:01-- this is very alarming. However, what concerns me is the specific Snort rule that triggered this, TFTP - Internal UDP connection to external tftp server, obviously this happens from the inside and the administrators want to monitor it--but why on Earth is this allowed? I took a look at 216.12.211.209 to see what I could figure out and simply tried to see if it had a web page. This is what was returned:

> This page is used to test the proper operation of the Apache Web server after it has been installed. If you can read this page, it means that the Apache Web server installed at this site is working properly.
>
> ---
>
> If you are the administrator of this website:
>
> You may now add content to this directory, and replace this page. Note that until you do so, people visiting your website will see this page, and not your content.

I had suspected this could be "legitimate" traffic after I saw the TFTP session, let me explain: sometimes people will use TFTP to send SSL certificates to the web server. Also, it's commonly used to send the configuration file, log, and icons onto a web server to complete installation. So, this could very well be a user trying to set up his own web page using IRC and TFTP to set it up. At any rate, we should investigate this system. Here's what ARIN reports about the destination IP:

```
OrgName: Everyones Internet, Inc.
OrgID:   EVRY
Address: 3333 Katy Frwy Houston TX 77007
Country: US
Comment:
RegDate:
Updated: 2000-02-24
```

Nslookup reports: topquark.roadkill.com
Here is what IANA reports on the source ports involved:
2613 smntubootstrap

2783 AISES 2783
This doesn't help me make any other conclusions. Let's move onto how else the MY.NET system is connecting to the other destination IP's. He used source port 3145 to 64.45.60.200 and port 3789 to 62.13.43.58. I noticed his system conducted a similar port switching to 66.250.105.173 like it did with the first IP. He started this connection on Oct 16 at 11:16 and stayed until 04:29 the next morning. Every time there was a new connection he started off on source port 2399 then switched to port 1989-- maybe that's normal IRC? Then, he came back on Oct 17 at 01:22 and stayed off and on throughout the day. Then he started up again on Oct 18 at 00:06 and stayed until 01:18--- does this guy ever sleep? I'd have to say something odd is happening with this system, or this guy likes IRC to communicate. It doesn't look automated because the times are not really consistent other than they seem to start mostly at night. Let's investigate this system for compromise.

24	130.85.91.240
18	130.85.84.178
10	130.85.87.42
10	130.85.150.213

I grouped these together because their alerts are all destined to port 65535 (Red Worm alerts again). I noticed that a lot of people confused this with the Code Red Worm in their papers-- this is not flagging on Code Red, the university set this up to really flag on the Adore rootkit, and simply searches for port 65535. I've already talked about this alert several times now in my analysis so I'm just going to analyze this for telltale signs that it really is the Adore worm (scanning class B subnets, scanning for ports 111, 515, 53, and 21, and if the system is sending traffic to port 25 -- the worm emails the attacker pertinent information like configurations and passwords). Thankfully, I didn't find any correlation or evidence of this. So, I next took a look at who they were going to, and then who else was going to them.
130.85.91.240 went to 131.156.182.149 and 137.45.71.61. The only other IP of interest going to them was *130.85.114.88.* We've already talked about this system in the external alerts section, so this solidifies this IP as one we should investigate. Additionally, 130.85.91.240 generated a ton of scan logs to all kinds of IP's and ports, and from all kinds of source ports. He was tagged with 45,898 separate destinations for UDP alerts, and 2,484 separate destinations for TCP alerts.
130.85.84.178 went to 2 destination IP's, 24.67.239.191 and 219.102.101.64. Additionally, we have 2 other MY.NET systems going there too, *130.85.83.146 and 130.85.70.176* (Again, we've already talked about them too, so we need to investigate these IP's). Also, this IP generated a bunch of scan logs looking for port 6257. In total, he scanned 5,744 separate destinations.

130.85.87.42 went to 1 destination IP-- What's unusual about this connection is the source port involved in every alert, 49305. This is definitely odd, and this IP needs to be investigated.
130.85.150.213 went to 1 destination IP, 68.0.161.156. One other MY.NET system went there too, *130.85.165.24*. We've already talked about this system too and it needs to be investigated. 130.85.150.213 also generated a bunch of scan logs, all looking for port 6257 and targeting 2,257 separate destinations. We need to investigate all 4 of these MY.NET systems for compromise and misuse.

Because I saw so many alerts concerning port 65535, I decided to look at MY.NET systems with this port as a source. Below I've listed these IP's:

33	130.85.140.9
3	130.85.188.24
24	130.85.6.40

130.85.140.9 went to 128.205.10.249 targeting all different destination ports in the 33K range. This shows me that we are really seeing return traffic most likely, and the MY.NET system is hosting something on port 65535. We need to investigate this system to see what service is bound to the port (again LSOF or Fport).
130.85.188.24 went to 10.0.1.1 and targeted port 192 (Now this is interesting in a lot of ways). First off, I now questioned the placement of the IDS sensor (how come it picked up this alert?). That led me to question the router configurations (why are they routing this private IP around?). Then, I wondered about the configuration of this private LAN (if it's leaking packets then it's not configured correctly as a multihoned system). Finally, we need to find the policy about hosting a separate LAN attached to the university's LAN. At any rate, let's try to figure out what's going on with this connection. Below I clipped out the portion of Snort_Snarf that captured this IP.

10/14-04:13:10.816777 [**] High port 65535 udp - possible Red Worm - traffic [**]
130.85.188.24:65535 -> 10.0.1.1:192

10/15-05:31:22.395664 [**] Port 55850 udp - Possible myserver activity - ref. 010313-1 [**]
130.85.90.118:55850 -> 10.0.1.1:192

10/16-19:19:46.507617 [**] Port 55850 udp - Possible myserver activity - ref. 010313-1 [**]
130.85.87.233:55850 -> 10.0.1.1:192

10/16-23:26:52.599642 [**] High port 65535 udp - possible Red Worm - traffic [**]
130.85.188.24:65535 -> 10.0.1.1:192

10/17-05:01:37.654297 [**] Port 55850 udp - Possible myserver activity - ref. 010313-1 [**]
130.85.188.24:55850 -> 10.0.1.1:192

We see that 2 other MY.NET systems went to the private IP and port, but they came in from port 55850 (flagged as 'Possible myserver activity'). But what is port 192? IANA reports this as the 'OSU Network Monitoring System', whatever that is but I'm pretty sure that's what's not running. Another possibility could be this is an Apple iBook looking for an AirPort base station via a discovery process over port 192. 10.0.1.1 is the default address given to an AirPort, and since Snort captured this log it shows me that the route isn't handled correctly within the private LAN and the traffic tried to route through the default gateway. The configuration program does the discovery by sending a broadcast UDP packet to port 192. The packet should contain 0x01, followed by 115 times 0x00, making a total of decimal 116 bytes (hex 74)--this is what we should look for to confirm this (I looked in the OOS files to see if we had captured this dump to analyze but no luck). The Base Station then responds with a UDP packet from port 192, back to the originating IP-number and port (which we don't see since we only captured one-way alerts). Once it gets the configuration, software updates are done over SNMP. It uses the community string 'public' and cannot be changed using the Apple software-- we don't see any evidence of this since a string of public' would certainly be flagged by the Snort rules.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP public
access udp"; content:"public"; reference:cve,CAN-2002-0012;
reference:cve,CAN-2002-0013; sid:1411; rev:2;
classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP public
access tcp"; flags:A+; content:"public"; reference:cve,CAN-2002-
0012; reference:cve,CAN-2002-0013; sid:1412; classtype:attempted-
recon; rev:4;)
```

I found only one other correlation for this type of behavior from the following posting at dbforums.com

From: Clvrmnky
Date: 09-11-02 16:46
Subject: Re: Advice with line in pflog, udp 192

Reply

clvrmnky <clvrmnky@coldmail.com.invalid> wrote in
news:Xns9284883C9A57Aclvrmnkycoldmailcomi@142.77.1.-194:

[...]
> I recently took a look at my latest /var/log/pflog, and noticed that there are
> literally hundreds of the following line:
> Sep 09 09:59:00.794164 rule 6/0(match): block out on xl1:
> xx.yy.aa.bb.52261 > 10.0.1.1.192: udp 4

Because this didn't generate a lot of alerts I probably would have missed this (and other annalists too) had I not performed my cross-reference methodology. I'm not totally convinced this is some AirPort base station broadcasting for

configurations, especially since we have known backdoor ports connecting to it. Also, I'm not totally convinced we have a private LAN either because of the very small amount of alerts to it. At any rate, we need to investigate all the MY.NET systems going to 10.0.1.1 and try to hunt down where the private LAN resides.

130.85.6.40 - I had briefly talked about this IP in the external alerts section but I dismissed it as a normal mail server. After looking at the internal alerts and ports I've come to a different conclusion. Below I've provided all the IP port to IP port combinations involved:

11 130.85.6.40 :25 64.83.112.10 :55850
7 130.85.6.40 :65535 209.132.220.171 :25
2 130.85.6.40 :25 64.75.37.159 :27374
1 130.85.6.40 :65535 208.35.99.86 :113
1 130.85.6.40 :65535 129.100.83.16 :113
1 130.85.6.40 :55850 63.251.244.194 :113
1 130.85.6.40 :41445 216.34.38.123 :25

What stands out is the second entry--this is what I was looking for in the first place, port 65535 to port 25. This seems to fit the pattern of the Adore rootkit and we need to investigate this system for sure. Additionally, we see this system connecting to Auth port 113, but I doubt it's really connecting to this service. I'd bet it's really the Invisible Identd Deamon, Kazimas. And hey, look, he's also connecting to Subseven (27374) and the MyServer Ddos agent. This system is most likely compromised.

15	130.85.88.230

Since I also saw 55850 quite a few times I searched on this port as both a destination and source. Aside from 130.85.6.40, 130.85.88.230 was really the only other MY.NET system going to port 55850. Below I listed the connections:

15 130.85.88.230 :1214 80.202.95.39 :55850

Since I've already talked about both the ports involved and what they could be I will just suggest we verify what's running on this system.
Below are the IP's that used 55850 as the source port:

39	130.85.104.66
230	130.85.86.106
1	130.85.90.118
1	130.85.87.233
3	130.85.188.24

Both **130.85.104.66 and 130.85.86.106** connected to the same destination using source port 55850, 239.255.255.253. All the connections went to port 427, so let's try to figure out what this service is about. Port 427 is used by several

different services; one could be NetWare 5 communication to locate servers (like Microsoft NetBIOS in a way). However, both the source and destination will be port 427 (again like NetBIOS). This is not what we see though, so I'm dismissing this as Netware traffic. Port 427 could also be used by HP Jetdirect devices to advertising their services. Some HP software utilities use multicast and port 427 to automatically discovery and automatically install a printer on the network. It's quite possible this could be happening, but to the destination IP listed? Let's take a look at what ARIN reports on this IP:

```
SLP Multicast [17] address, which is 239.255.255.253.  The
default
   TTL to use for multicast is 255.

      OrgName:    Internet Assigned Numbers Authority
      OrgID:      IANA

      NetRange:   224.0.0.0 - 239.255.255.255
      CIDR:       224.0.0.0/4
      NetName:    MCAST-NET
      NetHandle:  NET-224-0-0-0-1
      Parent:
      NetType:    IANA Special Use
      NameServer: FLAG.EP.NET
      NameServer: STRUL.STUPI.SE
      NameServer: NS.ISI.EDU
      NameServer: NIC.NEAR.NET
      Comment:    This block is reserved for special
      purposes.
             Please see RFC 3171 for additional information.
```

Okay, we can now dismiss this as HP traffic too. So what else could this be? This is what IANA reports port 427 as being: 'Server Location' -- well, what's this?

"The Service Location Protocol (SLP) provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet need little or no static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration."

SLP Requests messages are multicasted to The Administratively Scoped 239.255.255.253 address according to RFC2608. So, this seems to be what's going on here, so we have a false positive!

130.85.90.118, 130.85.87.233, and 130.85.188.24 were already discussed above when we talked about port 65535 and the private IP 10.0.1.1.

7 130.85.83.173

3 130.85.70.91
3 130.85.152.174

I grouped all of these MY.NET systems together since they all went to port 69, the Trivial File Transfer Protocol. This type of traffic should always be looked at since TFTP can be a dangerous tool or a transport to retrieve sensitive data. Let's take a look at just some of what TFTP can be used for. Basically, TFTP provides remote access to files, without asking for a password. It is typically used for the initialization of diskless computers, X terminals, or saving/uploading router configurations. However, if someone is hosting a TFTP server an attacker can connect to this service and possibly retrieve any other file accessible through the TFTP server. If binary files/programs are accessible, the attacker can analyze this to learn about the operating system and hardware used. If configuration files are accessible, analysis might reveal sensitive information--remember, there is no authentication involved; you will have the rights of the TFTP service. Even better the attacker might be able to traverse and grab the /etc/passwd file on vulnerable UNIX systems or any other file accessible through the TFTP server. TFTP can also be used as a DoS agent against some Cisco IOS's and a transport for unauthorized access to ADSL modems-- and so on. Simply, TFTP should be investigated and eliminated. Fortunately, none of our MY.NET systems seem to be using TFTP as a source port, but since we have them accessing external TFTP systems we have to conclude they are compromised and sensitive data is being uploaded. Another possibility is maybe the users of the internal systems are uploading router configuration files, it's quite possible if the university is using ATT and Sprint ISP links based on the ARIN output below, I'll try to traceroute now! --> Nope, that doesn't seem to be it. The bottom line however, is we need to investigate these.

216.22.147.226

```
9 Net Avenue, Inc. NINENETAVE (NET-216-22-128-0-1)
                                  216.22.128.0 -
216.22.255.255
Cobra Networks NNA-216-22-147-2 (NET-216-22-147-2-1)
                                  216.22.147.2 -
216.22.147.252
```

12.249.202.225

```
OrgName:    AT&T WorldNet Services
OrgID:      ATTW

NetRange:   12.0.0.0 - 12.255.255.255
CIDR:       12.0.0.0/8
NetName:    ATT
NetHandle:  NET-12-0-0-0-1
```

204.215.181.6

```
OrgName:    Sprint
```

```
OrgID:      SPRN

NetRange:   204.212.0.0 - 204.215.255.255
CIDR:       204.212.0.0/14
NetName:    SPRINT-BLKB2
NetHandle:  NET-204-212-0-0-1
Parent:     NET-204-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1-AUTH.SPRINTLINK.NET
NameServer: NS2-AUTH.SPRINTLINK.NET
NameServer: NS3-AUTH.SPRINTLINK.NET
```

Possible Trojan server activity

This is a rule the university set up to look for at least port 27374, the SubSeven trojan. I provided all the MY.NET IP's that either went to this port or used this port.

13 130.85.25.21 :143 4.35.54.173 :27374
3 130.85.113.4 :1214 194.209.15.189 :27374
2 130.85.83.201 :6346 213.177.137.33 :27374
2 130.85.6.40 :25 64.75.37.159 :27374
1 130.85.116.68 :7625 138.16.135.1 :27374
1 130.85.105.42 :1726 207.192.130.188 :27374

Looking at the first IP we could be concerned with either port involved, 27374 or 143 (the IMAP port). IMAP (Internet Message Access Protocol) is simply a protocol for e-mail clients to retrieve and work with e-mail messages on a mail server. I won't go into all the vulnerabilities and security holes involved with IMAP (like buffer overflows, obtaining super user accounts, DoS) but I'll provide some links that discuss this:
http://www.scanalert.com/news/16.jsp
http://online.securityfocus.com/advisories/2179
I did an nslookup of the destination IP however it didn't help me in determining if this is really IMAP traffic:
washdc3-ar1-4-35-054-173.washdc3.elnk.dsl.genuity.net [4.35.54.173]
We should investigate this IP for compromise.
130.85.113.4 is using the Kazaa default port we've already talked about. This doesn't prove it's Kazaa or subseven, so we should investigate this IP too.
130.85.83.201 is using the gnutella default port, but again this doesn't prove a thing-- we should investigate this too.
130.85.6.40 is already compromised, we know this and here is more evidence of that.
130.85.116.68 is using a source port I'm not familiar with (maybe it's just ephemeral)-- if so, then we should really be concerned since it's most likely going to a subseven system. We should investigate this for compromise and to see if any service is bound to port 7625.

130.85.105.42 is using a source port of 1726. I'm not sure what this is used for (maybe it's ephemeral, I think I may have seen this as a game port too) but we need to investigate this too for compromise. Also, this IP generated a lot of TCP scan alerts (238) to a bunch of separate IP's and destination ports-- that's not a good sign at all.

Why didn't I label the following IP's as concern?

130.85.53.36 -- Looks like web traffic to 218.55.184.152 (Asia Pacific Network) although a lot of other MY.NET IP's going to it--- This could be a diversion to some backdoor or automated scanner using port 80 to fool us it's web, but since it flagged on HTTP alerts written to look at payload content I would assume it really is web traffic--oddly I can't get to it via HTTP. Keep an eye out for these IP's, since it looks like the Asian IP is compromised.

The IP's below belong to Netscape and Cobalt, I looked at this and it appears to be normal web traffic.
207.200.86.97
207.200.86.66
216.241.219.14

130.85.153.146: This IP came up quite a bit (604 alerts) when I was searching port destinations. Although almost all of the destinations are in the 211 network range (Asia Pacific Network Information Centre), they are valid web sites--- this person is most likely an exchange student, or speaks some Asian language.

Scans: External

Total: 1404218
From outside: 158602

I only looked at the external scans to try and correlate my top IP's with it and to find any reconnaissance--otherwise scanning is going to happen daily and we wanted to concentrate on compromises. Below I've provided quite a few MY.NET IP's (most we've already identified or are suspected of compromise) as the destination for UDP, TCP SYN, and 'other' scans originating from the outside. This will be just a reference for internal IP's that should be earmarked as targeted and possibly vulnerable. Although I'm not going to analyze these alerts we need to treat them with importance. I would keep an eye out for unusual activity to or from these, make Snort rules to trigger on them, and run vulnerability assessments on them.

UDP

60 130.85.112.204
3 130.85.132.23

Other Scans

18 130.85.91.240	3 130.85.114.88
15 130.85.70.176	2 130.85.88.243
11 130.85.113.4	2 130.85.84.178
8 130.85.83.201	2 130.85.83.94
3 130.85.88.165	2 130.85.198.204
3 130.85.83.146	1 130.85.114.45

TCP

35 130.85.100.220	15 130.85.114.88	7 130.85.140.90
33 130.85.132.23	14 130.85.88.243	6 130.85.87.42
26 130.85.113.47	14 130.85.70.50	6 130.85.70.91
26 130.85.113.43	13 130.85.198.204	6 130.85.140.95
25 130.85.113.45	13 130.85.113.4	6 130.85.116.68
24 130.85.113.40	12 130.85.100.158	6 130.85.112.204
22 130.85.70.176	11 130.85.84.178	5 130.85.91.240
22 130.85.113.49	11 130.85.105.42	5 130.85.163.132
21 130.85.70.173	10 130.85.87.50	3 130.85.150.213
20 130.85.113.44	10 130.85.139.10	3 130.85.104.66
20 130.85.113.42	10 130.85.114.45	3 130.85.80.144
19 130.85.70.103	10 130.85.104.204	2 130.85.83.94
19 130.85.184.178	9 130.85.88.165	2 130.85.165.24
19 130.85.113.41	9 130.85.140.9	2 130.85.150.216
18 130.85.113.46	8 130.85.87.233	5 130.85.91.240
17 130.85.113.48	8 130.85.83.146	5 130.85.163.132

Scans: Internal

Total: 1404218
From inside: 1245616

Top Talkers

I decided to focus my scan analysis on MY.NET systems as a source of scanning, because this would indicate compromise or misuse. Below is the list of IP's I narrowed down to investigate; it is a mixed list of top IP's by number and correlation with some IP's we have identified in the alert sections.

130.85.114.88: This IP had 65,306 UDP scan alerts, originating all from source port 2939. It went to 20,296 separate destinations and scanned 3,404 different ports (most targeted destination port 1214 (Kazaa-- maybe this is scanning for the Morpheus exploit). However, I would bet these alerts are not caused from the internal system scanning away, but responding to outside IP's trying to connect to port 2939 (whatever is being hosted there, I would guess it's a file-sharing application). If we cannot verify a legitimate service bound to this port then we would conclude this is a compromised system or a user using an automated scanner. Further, this system had 243 TCP alerts going to 172 destinations and 125 separate ports (most to 80 and 1214), from random high-number source ports. At first I thought maybe the port 80 logs were just web returns but it was unusual that each separate destination IP received only one alert, plus I went to a few of the IP's and they are not serving a web page (maybe it's infected with NIMDA or CodeRed or some variant searching for other vulnerable hosts). This system was identified in the external alerts section under the Watchlist 000220 IL-ISDNNET-990517 heading.

130.85.88.243: This IP had 71,483 UDP alerts all from source port 1214. It went to 16,392 destinations and 3698 separate destination ports. I would conclude this too is actually return traffic to outside IP's trying to connect or scan for port 1214. This system also had 333 TCP alerts going to 187 destinations and 150 separate destination ports (again mostly port 80 and 1214). I would say this is probably return traffic too but I noticed a lot of odd ephemeral port pairs. This IP was identified in the external alerts section and needs to be investigated.

130.85.83.173: All of the UDP alerts for this IP (80,595) went to 1 destination; 216.22.147.226. This destination IP was already identified in the internal alerts section when I was discussing the TFTP transfers to it, so now I'm pretty confident we have an issue here. The MY.NET system went to 30,012 separate ports to the destination IP from 3,971 different source ports (the neat thing I noticed are sequential ports from 5000 all the way down to about 4000, in perfect order, and each port generating between 20 to 15 alerts as they scanned a separate destination port-- odd, this is definitely automated). Additionally, this IP had 37 TCP alerts all going to port 6667 (Perhaps ScheduleAgent, Trinity, WinSatan) mostly to 1 IP, 128.211.244.150 (hltp-c-150.resnet.purdue.edu). We need to investigate this system for compromise.

130.85.198.204: This IP generated 64,890 UDP alerts and went to 10,883 separate IP's and 3,586 different destination ports. Again I saw the familiar source port used in every alert, 1214-- so I'm guessing this is return traffic to systems scanning for either the Kazaa application or the Morpheus exploit. Also, this IP had 86 TCP alerts. It went to 62 different destinations, 54 separate ports, and came from 68 different source ports (most in the 3k range but nothing stuck out as logical). We need to take a look at this system.

130.85.87.50: This IP had 48,630 UDP alerts targeting 3,060 destinations and 3,635 destination ports (by far, port 27005 was targeted the most). The odd thing about this is we only have 4 source ports involved (mostly port 888 and 999). Now, these are valid, registered ports with IANA and plenty of applications use these ports even if we don't come across them daily (yeah, and so do Trojans!)-- and even port 27005 is used in gaming. But, do we really have legitimate traffic here? I really can't see this as okay traffic since it involves so many destinations, and it's not the Half-Life game port either since the server most of the time listens on port 27015. Further, it just seems too odd to go from port 888 to 999. Let's investigate this system. Also, there were 3 additional TCP alerts from source port 2309 to port 2300. I don't know what this transaction is about but I'd guess it's so kind of gaming port since a lot of games register in this range (but then, wouldn't we see a bunch of other alerts?). Let's take a look at what's bound to this port.

130.85.139.10: All 46,664 alerts generated by this IP came from source port 1906. It went to 13,067 different destinations and to 3032 separate ports. This one can be interpreted in 2 ways; either a lot of people are scanning for port 1906 (but then where are the other MY.NET IP's if it's a general scan for this port?) or this is return traffic from something running on this port (this seems more likely, what we have is probably an automated scanner). This IP also had 603 TCP alerts targeting 302 destinations and 252 separate ports (most to port 1214). The system used 480 different source ports that incremented sequentially but not right in order. This looks like a normal session with a Kazaa client.

130.85.88.165: We had talked about this IP in the external alerts section when I analyzed a Watchlist IP connecting to port 4509. Now we see this IP has 250 UDP alerts to 248 destination and 234 separate ports. Again, we see a common theme where the source port remains constant at 1214-- we still need to check it out though. This IP also had 5 TCP alerts all being a different source-destination port pair... I don't see any logical connections except the source ports increment sequentially. I don't have an explanation for what he's doing via TCP but we should plan on monitoring this system.

130.85.104.204: This IP generated 76 UDP alerts targeting 74 destinations and 74 ports. All the alerts came from source port 1214 (so we have return traffic most likely from outside users accessing the Kazaa port). This IP also has 4,697 TCP alerts targeting 4,693 destinations (all in the same subnet). All the alerts came from 2,870 source ports to only 3 destination ports --mostly port 4523. This is a very odd port (it's not registered with any application or games I know of). Perhaps it could be the Celine virus posting a backdoor-- but it's supposed to run on port 4523 not scan to it (I think). What I assume is going on is an automated scanner set up to search for port 4523 for some reason (perhaps to access the Celine backdoor). All of the destination IP's are in the 128.228 subnet. We need to investigate this system for misuse.

130.85.6.40: We've talked about this IP several times already so it's not a surprise we see it in the scan logs too. It generated 1848 TCP alerts and went to 1040 different destinations. It's source ports were all in the 30K to 50K range and they targeted port 113 (1521 times) and 25 (327 times). When I first saw this

I thought the system might be searching for the sendmail exploit: There is a buffer overflow in version 8.6.9 of sendmail when it processes the ident service responses. Sendmail makes a connection to the ident service on the client host in order to log information about the user who is making the connection. However, an attacker could send a very long response instead of what's normally expected causing a buffer overflow of that service. The attacker then execute arbitrary commands as root on the server. This vulnerability was described in an X-Force alert and is posted at CVE 1999-0204. Since the 2 services go hand in hand for the exploit, and thus the ports, I searched each destination IP to see if it was scanned for both ports-- while some did have this in common not all of them matched like I expected. In doing this analysis I noticed that the MY.NET system seemed to be concentrating on several classes of networks (mostly 66, 209, 200, 216, 64, 210, 205, 152, and the 12 networks). If could quite possibly be legitimate returned traffic toan outside mail server, but since so many destinations are involved this is highly unlikely. If this isn't the sendmail exploit it is something abnormal, and since we have already identified this system as compromised we can assume this scanning is malicious in nature.

130.85.150.216: This IP had 982 TCP alerts targeting 638 destinations for mostly port 445 and 139. Since the source ports were changing sequentially I know this system is scanning for SMB and NetBIOS services. Additionally, he targeted only 3 separate networks (130.237, 24.180, and 134.53). We need to investigate this IP for misuse.

130.85.150.214: This system generated 175 UDP alerts targeting only 1 destination, 224.0.1.22 --hmmm, that looked familiar so I did a whois on it and it's actually a reserved IP used for multicasts. I took a look at the source port to destination port pairs on a hunch they would be the same, and they were both using port 427. I already talked about port 427 in the internal alerts section and this appears to be a Netware5 system using a multicast against it's subnet to find it's default gateway because it's misconfigured. You can read RFC 3171 for more info but basically if the client does not have a default gateway (and thus routes) it doesn't know where to go, so it send this packet destined to the reserved IP that routers understand in order for them to help supply a default route (I hope I explained that correctly). Also, this system generated 963 TCP alerts going to 620 different destinations. Just like the IP above, this system target mostly ports 445 and 139. This time, it concentrated on 6 separate networks (35.11, 134.71, 24.188, 152.3, and 12.241). We need to investigate this system for misconfiguration and misuse.

130.85.132.23: This system had 910 TCP alerts all going to port 445. He targeted 827 different IP's using sequential, steady ports and concentrating mostly on the following networks; 35.11, 160.252, 160.10, 134.76, 24.186, 170.28, 152.3, and 134.193)-- this is definitely a deliberate scan and should be investigated.

130.85.163.132: This system had 476 TCP alerts all going to 1 system, 128.183.252.83. There were also 476 source ports used, all within the 33K-35K range targeting 457 separate destination ports in the following ranges; 5200-5400, 17,600-17,800, and 1243-1420. The system also targeted port 21

separately. I really don't know what's happening with this but we should investigate what the system is doing and try to determine who initiated the traffic. The destination IP belongs to gsfc.nasa.gov, Goddard Space Flight Center in Maryland, and it hosts an RSD Portal web page-- perhaps there are some programs or streaming demo applications causing these alerts?
130.85.114.45: Our final IP generated 28,074 UDP alerts all using source port 2917 and going to 2,825 separate destination ports. This obviously appears the MY.NET system is hosting something on this port, and we are seeing return traffic-- it is possible that the MY.NET system is using an automated scanner or packet crafting to scan outside systems but the data just doesn't fit this. First, the data appears to be 'normal' ephemeral port ranges and second the system went to 10,194 different destinations--- if he is scanning, why to so many IP's to only 1 or 2 ports apiece? We need to determine what is bound to port 2917 and why so many outside IP's are interested in it-- or if this is indeed a scanner set to source port 2917 we need to determine this too. There were also 345 TCP alerts targeting 207 separate destinations and 181 ports. The system used 275 different source ports ranging all over in the low thousands. The destination ports seemed to have no pattern or logic either. In other words, I'm not sure what these odd ephemeral port pairs are about, so let's tag this IP as one we want to monitor.
A lot of other IP's were not identified within this section because of the amount of false positives involved--- here are some general observations of what we can do to limit these in the future. Perhaps we could utilize the portscan-ignorehosts switch in the scan preprocessor to reflect some of the IP's that are generating a lot of alerts but seem to be legitimate traffic (if our course the policy set by the university will allow such streaming media and file-sharing applications). Additionally, a lot of alerts were generated because of the presence of SYN flags within the packets. If an internal system matched the threshold of the scan preprocessor that looks for SYN scans then all the traffic for that session is captured. More than likely the portscan switch is set to the default value of 4 connection attempts within 3 seconds, so obviously a lot of normal web traffic will be flagged here. We should fine-tune this preprocessor to either flag on less time between SYN's or a higher number of connection attempts per second.

OOS Analysis:

The OOS alerts were broken up into both external and internal events (just like I did for the alerts and scans). I concentrated on the external OOS alerts only as checks for correlation and verification with what I suspected systems were doing. Internal OOS alerts were analyzed as yet another means for finding anomalies or possible compromises, and also to help correlate any MY.NET IP's I already identified-- luckily, only 2 MY.NET systems were generating alerts from the inside.
130.85.12.4 - This IP looks like it's getting scanned for IMAP and IMAP secure ports (143 and 993) and sending Reset packets back to the initiator. But the odd thing is they are all coming from 10 different MY.NET systems. So the question

is why are these IP's trying to access 130.85.12.4? Below I've provided a snip of 2 of these traces:

10/14-11:11:56.656137 130.85.12.4:993 -> 130.85.177.60:1118
TCP TTL:255 TOS:0x0 ID:62675 IpLen:20 DgmLen:40
12***R** Seq: 0x1ABC0715 Ack: 0x0 Win: 0x0 TcpLen: 20

10/14-14:47:32.360883 130.85.12.4:143 -> 130.85.168.39:51181
TCP TTL:255 TOS:0x0 ID:12985 IpLen:20 DgmLen:40
12***R** Seq: 0x51962DE0 Ack: 0x0 Win: 0x0 TcpLen: 20

130.85.70.183 -> 130.85.1.4: This IP pair generated 389 alerts, almost all of the OOS internal alerts logged. I looked at this traffic, for a while, and it appears 130.85.70.183 is doing a time broadcast over TCP-- I'm guessing it's an NTS box (TTL of 64, Window size 1452-- seems odd though) broadcasting a time request to the time node 130.85.1.4. Although it appears 'normal' since the source ports and IP ID increment almost sequentially, it definitely needs to be checked for how it broadcasts. The weird things about the trace, however, are no TCP flags are set and whenever it does it's burst of connections at some time period the sequence numbers are always the same. Here is a snip of the traces:

10/14-16:03:31.632702 130.85.70.183:55131 -> 130.85.1.4:37
TCP TTL:64 TOS:0x0 ID:3031 IpLen:20 DgmLen:40
******** Seq: 0x84000000 Ack: 0x0 Win: 0x5AC TcpLen: 20

10/14-16:03:33.629558 130.85.70.183:55131 -> 130.85.1.4:37
TCP TTL:64 TOS:0x0 ID:3032 IpLen:20 DgmLen:40
******** Seq: 0x84000000 Ack: 0x0 Win: 0x5AC TcpLen: 20

I would chalk both of these IP's as 'network issues' for a later study, but not necessarily compromises or misuse.

External Source Address Chosen For Investigation: A Summary

Here is the list of IP's I decided to investigate based on the alerts they generated and their relationship to other IP's and internal systems--they have all been discussed throughout the paper, click on the IP to go to their discussion.

212.179.103.7 *Click the IP to go to the section where I analyzed it*

cablep-179-103-7.cablep.bezeqint.net [212.179.103.7]

RIPE.net:

inetnum:	212.179.100.0 - 212.179.124.255
netname:	CABLES-CONNECTION
mnt-by:	INET-MGR
descr:	CABLES-CUSTOMERS-CONNECTION
country:	IL

```
admin-c:      MR916-RIPE
tech-c:       ZV140-RIPE
status:       ASSIGNED PA
```

137.45.71.61 *Click the IP to go to the section where I analyzed it*

dhcp-71-61.radford.edu

```
      OrgName:    Radford University
      OrgID:      RADFOR

      NetRange:   137.45.0.0 - 137.45.255.255
      CIDR:       137.45.0.0/16
      NetName:    RU-NET
      NetHandle:  NET-137-45-0-0-1
      Parent:     NET-137-0-0-0-0
```

202.102.233.93 *Click the IP to go to the section where I analyzed it*

```
inetnum:      202.102.224.0 - 202.102.255.255
netname:      CHINANET-HA
descr:        CHINANET Henan province network
descr:        Data Communication Division
descr:        China Telecom
country:      CN
admin-c:      CH93-AP
tech-c:       LZ33-AP
mnt-by:       MAINT-CHINANET
mnt-lower:    MAINT-CHINANET-HA
```

80.137.155.155 *Click the IP to go to the section where I analyzed it*

p50899B9B.dip.t-dialin.net
RIPE.net::

```
inetnum:      80.128.0.0 - 80.146.159.255
netname:      DTAG-DIAL16
descr:        Deutsche Telekom AG
country:      DE
admin-c:      DTIP-RIPE
tech-c:       ST5359-RIPE
status:       ASSIGNED PA
```

130.230.20.245 *Click the IP to go to the section where I analyzed it*

ftekpc72.ce.tut.fi

```
      OrgName:    Tampere University of Technology
      OrgID:      TUT

      NetRange:   130.230.0.0 - 130.230.255.255
      CIDR:       130.230.0.0/16
      NetName:    TAMNET
      NetHandle:  NET-130-230-0-0-1
Parent:     NET-130-0-0-0-0
```

Internal Systems Identified As Possibly Compromised

Here is the summary of MY.NET systems we should investigate for compromise, misuse, and network issues. Also listed are IP's we should earmark for additional monitoring. All IP's have been discussed already, so they are hyperlinks to their portion in the paper.

130.85.70.91	130.85.83.146 130.85.70.176 130.85.165.24	130.85.150.216
130.85.91.240 130.85.88.165 130.85.114.88 130.85.112.204	130.85.140.9 130.85.188.24 130.85.6.40	130.85.25.21 130.85.113.4 130.85.83.201 130.85.116.68 130.85.105.42
130.85.83.94	130.85.100.220	130.85.198.204
130.85.100.158	130.85.88.230	130.85.87.50
130.85.70.103 130.85.70.50	130.85.84.178 130.85.87.42 130.85.150.213	130.85.139.10
130.85.168.92 130.85.88.243	130.85.90.118 130.85.87.233	130.85.104.204
130.85.80.144	130.85.83.173 130.85.70.91 130.85.152.174	
130.85.150.214	130.85.132.13	

Defensive Recommendations

My defense recommendations will be an extension of what I presented in the Executive Summary section, not a concentration on cleaning up specific alerts or network issues-- that's what a network and vulnerability scanner can provide us. Even though I provided links to specific exploits and vulnerabilities (and the alerts themselves often have reference links) which will help us understand and fix the vulnerabilities, my concentration is on identifying what's going on with the network-- and the ultimate goal is to be able to provide enough information to help define network policy, defense actions, and incident response.

FOR STARTERS: The very first thing I would do is assemble an incident response team or vulnerability assessment team to investigate the suspected compromise list I provided, and to provide their own penetration and risk assessment for the network. What needs to be done is a vulnerability and identification scan on the internal network to identify security holes and to validate services running on either known servers (HTTP, FTP, Mail, DNS, etc.).

IP BLOCKS, TOP PORTS ATTACKED, FBI TOP 20, SECURING ROUTERS: The next step would be to understand and implement the headings above. First off, we can get a list of networks to block at: http://feeds.dshield.org/block.txt. Similarly, we can set up a block list for the top 10 IP offenders located at http://www.dshield.org/top10.html . Dshield also provides a list of the top ports scanned for or attacked, this will help us in our decision on what to allow or deny in our network and give us an idea of what to keep up on with security patches: http://www.dshield.org/topports.html. Not surprisingly, the FBI top 20 vulnerabilities relates a lot to what ports are being scanned. Below is the list of the vulnerabilities; you can access this site for an explanation of them: http://www.sans.org/top20/. Basically, the FBI top 20 are common, well-known vulnerabilities. The reason why they are still effective in most attacks is based on tools widely available to exploit them and the fact administrators either don't fix them or forget to fix them on rebuilt systems.

Top Vulnerabilities to Windows Systems

- W1 Internet Information Services (IIS)
- W2 Microsoft Data Access Components (MDAC) -- Remote Data Services
- W3 Microsoft SQL Server
- W4 NETBIOS -- Unprotected Windows Networking Shares
- W5 Anonymous Logon -- Null Sessions
- W6 LAN Manager Authentication -- Weak LM Hashing
- W7 General Windows Authentication -- Accounts with No Passwords or Weak Passwords
- W8 Internet Explorer
- W9 Remote Registry Access
- W10 Windows Scripting Host

Top Vulnerabilities to Unix Systems

- U1 Remote Procedure Calls (RPC)
- U2 Apache Web Server
- U3 Secure Shell (SSH)
- U4 Simple Network Management Protocol (SNMP)
- U5 File Transfer Protocol (FTP)
- U6 R-Services -- Trust Relationships
- U7 Line Printer Daemon (LPD)
- U8 Sendmail
- U9 BIND/DNS
- U10 General Unix Authentication -- Accounts with No Passwords or Weak Passwords

The final thing is to secure the routers against overflows, DoS, vulnerable services (like SNMP for example), etc. Cisco provides a pretty good write-up for general router security, design, and administration at http://www.cisco.com/warp/public/707/21.html.

SOFTWARE UPDATES/PATCHES Install the latest software and security patches for both the OS and the services that allow network connections. Policy will dictate when and how often patches and hotfixes will be applied, but no policy should ignore this.

INVENTORY/BENCHMARK: This is related a lot to what an assessment team would produce after their scanning is complete. Basically, we want to make an

inventory of the network to establish what systems are out there and what services are accepting connections. This helps us define acceptable use and what we need to secure. We know from the Top 20 vulnerabilities that numerous services are either on by default in clean installs or the administrators forget to patch them, this inventory should help identify this too and our policy.
POLICY: I already talked about the Snort rules the university set up to monitor certain activities acknowledged as dangerous, but still allowed to be run on the network-- we need to come up with a better solution and a better policy on security and acceptable use. Below are just a few issues that the overall network policy should address, http://www.sans.org/resources/policies/ provides some additional guidelines and templates for all kinds or policies.

- **INCIDENT HANDLING/RESPONSE** An incident handling policy should be established for preventive measures and response to suspected compromises. Within this, Incident groups should be defined. For instance, multi-tiered handling groups can be established for each stage within the incident handling process. The first wave could be technical advisors consisting of firewall administrators, system administrators, and information assurance specialists. The second tier could be members of the Network Specialty Operations Center (NSOC) that act as the medium link between the technical staff and outside agencies such as law enforcement and media. The third group consists of the CIO's and legal offices that ultimately receive the data collection and recommendations from the previous groups. This group defines both containment and remedial action for the technical group to accomplish, and provides direction for the eradication methods.
- **ANTIVIRUS** Virus solutions need to be implemented throughout the network. For instance, all mail servers should implement virus scanners that do an inspection and forward method of delivery. For instance, implementing a global antivirus server providing application level filtering and scanning is a pretty good step in the first line defense against Trojans or malicious code from entering the network. Similarly, we should install host-based AV programs that the global AV servers can push signatures to.
- **PERIMETER** We need to establish a perimeter, a demarcation, for the network. Like I stated in the executive summary section, their does not appear to be firewall or router ACL's in place filtering IP's, ports, or protocols. Further, there does not appear to be a clear distinction between server and client placement within the network. Finally, I don't see any evidence of a DMZ implemented to store servers that would offer services to the outside world. Even if implementing a firewall is too costly or against the university policy to provide open research, we can at least take care of the last two observation with router segmentation, or subnetting, and locking down these subnets. Egress and Ingress filtering could be established to dictate what IP's can access those subnets and what the systems on the subnet can do and where they can access.
- **INGRESS/ EGRESS** Aside from conducting this on or demarcation, we should apply this to our perimeter router as well. Like I stated above, we should block the recommended subnets provided at DSheid and the top

offending IP's but also we should configure the router to reject IP's coming into the network that have the same IP addresses as our internal systems. Similarly, we should prevent traffic from leaving the network that do not have an IP within the internal network.

- **VULNERABILITY WATCHERS** Here we have watchers that research and gather current intrusion trends and vulnerabilities, as well as tracking incidents and application bugs. Once the watcher collects what is pertinent to the network and software, they research how to rectify the problems.
- **BANNERS ON ALL SERVICES** This goes beyond just the log-on banners. Every service provided by an internal host, or DMZ host, has to have a banner wrapper warming the user that the system should not be accesses without proper authority and their actions will be monitored.
- **ROUTINE SCANS AND AUDITS** Utilizing such programs as Nessus, Snort, and ISS can go a long way in identifying vulnerable systems and services, and help us identify new systems popping up on the network too. Port Monitoring can also help identify a benchmark and suspicious services, tools like Nmap scans can report if ports are in use. System file Monitoring like Tripwire and Winalysis can detect changes in files, the registry, user accounts, rights policy, services, shared drives, and volumes. They both can detect intentional tampering, user error, software failure, and introductions of malicious software, as well as open doors. Port-to-Application Monitoring is another method to help us identify what's going on with a system. Tools like FPort and LSOF report all open TCP/IP and UDP ports and maps them to the owning application. Utilizing All-in-Ones Scanners like SamSpade, ManiacScanner, and WS PingPro can help administrators with basic analysis such as pinging, lookups, who-is, port scans, OS fingerprinting, traceroute, NTP, SNMP, and others. Implementing sniffers like Analyzer, Ethereal, and TCPDump can help us capture and breakdown packets on the network for dissection or troubleshooting. Finally, it would be a good idea to have some sort of a Port/Service integrity check. For instance, we have a program that ports Nmap scans using the –O –sS option and compares services and ports with its' database previous scans and emails us a report of the differences. It also checks for rouge IP's (IP's grabbed out of thin air without being properly registered) against the list of valid and registered user IP's.
- **SECURITY AWARENESS TRAINING AND EVALUATION (SATE) TRAINING** Perhaps we can institute a standard and mandatory training session for all users on the network. Some examples of SATE training might include what to do if you receive an application embedded within an email, how to identify and annotate social-engineering hacker attempts, and how to identify what services your computer should be running. There could be many other informational modules the user learns like password storing policies, Internet policies, data storage and destruction, and so on.
- **FULL SYSTEM BACK-UPS** This should be conducted on critical servers, and done at least bi-weekly. Additional policies for how to store the images should be created as well, but at least stored off site in a safeguard room.

CONTINUED ANALYSIS What I conducted for this week should be a reoccurring process to help ensure good security practices and policies. Uur defensive policies will continue to evolve and reflect changes in our network and changes in attack patterns. Again, this will help give us a baseline, or benchmark, of what may be normal traffic and what attackers are targeting, and ultimately how we should go about setting up defense recommendations. Another step in this could be funding annual security visits to access information assurance against a standard set of guidelines.

Description of Analysis Process

In the section Analysis: Preface to Methodology , I fully detailed how I analyzed the data, came to my conclusions, and prioritized the alerts. So, below I'll only talk about what I did to create manageable data and what tools I used to manipulate the raw log files.

CONCATENATING THE FILES: Once I downloaded the files for alerts, scans, and OOS, I merged them together to produce a single file for each category. For example, I would issue the following command to copy all alert files into a single file called alert.txt--- this is issued on a Windows system:

```
copy  Alert*.txt Alert.txt
```

CLEANING UP THE FILES: This step had several different methods depending upon the output I wanted to analyze. I'll discuss how I broke out each type of file

Alert file: The first thing I did was tried to figure out what IP octets MY.NET represented. I happened to find the octets (130.85) that the university was running when I was browsing through some web packet dumps in the OOS files-- I saw the initial IP address and user agent. So, I then tried to verify that this indeed represented the internal systems by looking at the alerts for these octets. I noticed some rules like, "TFTP to internal help desk system 130.85.83.197", so I knew I was right on. Most people in their papers ran into problems running Snort_Snarf because it didn't understand the MY.NET substitution for an IP, so they decided to replace MY.NET with some private IP range or an IP that wasn't logged in any of the alerts-- this can lead to problems since the real MY.NET IP will be flagged as an external alert in some cases. The next thing I did was got rid of all the spp_portscan entries in the file. This was a basic grep command:

```
grep -v "ssp_portscan" alert.txt > alertfull.txt
```

. I didn't want the ssp entries in my alert analysis since this data was already included in the scan logs. Next, I wanted to separate the external alerts from the internal alerts, basically I greped out any instance of 130.85.x.x after the direction arrow. For instance, issuing this command will look for all instances of 130.85.x.x as a destination and redirect the output into a file called alertexternal.txt:

```
cat alertfull.txt | egrep "\-\>.*MY.NET" > alertexternal.txt
```

Notice how you have to identify the direction arrow using grep commands. Because the dash is actually used in expressions you have to place the slashes around it to tell grep to search for the dash and not use it as a command. The dot after the arrow tells grep to look at everything past this, including spaces until the MY.NET string is found. For the internal alerts I used the same command but with the -v switch, this will give me all alerts with MY.NET as the source. Next, I

replaced every instance of MY.NET in both of the files with 130.85. Since I'm not much of a Unix editor person, I used a GUI program called Eluent Tools, specifically the Replace program. I tested this tool with a few test files and it seemed to be error free, and super fast. All I did was tell it what to find and what to replace it with, and it created a new alert file in about 2 seconds. So, now I have 2 new master files to work with, one internal and one external. Now I was ready for the first part in my analysis, running these through Snort_Snarf. For the Snort_Snarf output I left the separate files as is (meaning their original formats) and piped it through the perl script. I issued the following command to log it to the directory 'output', use the Windows platform, reverse the order, and list the top 10 IP's rather than the default of 20: perl snortsnarf.pl -d f:\output -win -rs -top=10 alertfull.txt. I also decided to pipe it through another perl script called Snort_Stat that analyzes the data strictly by numbers and statistics. I issued the following command:
type f:\datadumps\alert\alertinternal..txt | perl snort_stat.pl -h> alertinternal.html
Basically, you must first have the file read to standard output by issuing the 'type' command, or 'cat' if you prefer. Then you pipe that output to perl so you can run the snort_stat script. The -h option tells it to format the output in HTML so that's why I redirected it to the file called alertinternal.html. Next, I wanted to analyze all the data according to IP pairs and port pairs like I described in the methodology section. To do this there are a lot of ways like using sed, VB, perl, or AWK. I found a nifty perl script that was pretty close to what I wanted, and a much faster way of getting the output, called process_scanalert.pl-- but I'm not sure who originally wrote it. The program gives you the ability to print the source or destination IP's or their pairs (and counts them too), destination ports, fingerprinting attempts, and even an spp_portscan summary. For example, to find the targets that were attacked you would issue this command:
perl -s process_scanalert.pl -o alertinternal.txt > alertinternalports.txt
You need the -s switch since this script is written to handle perl arguments at the command line-- I've provided the source code in the appendix. What's really cool with this too is it will handle the file 'as is', so you don't need to format it into some other delimitated fashion. However, in the end I decided to just use AWK for my needs but now I had to clean up the files even more. Next I turned back to the Eluent replace tool to find any instance of "[**]" and replace it with &. The '&' symbol will now be used to delimitate the files. Also, I searched for "->" and replaced it with the '&' symbol too. So, below is the original log file:
10/15-00:02:31.847188 [**] SMB Name Wildcard [**] 200.216.167.81:1025 -> 130.85.133.252:137
And I made it look like this in the end:
10/15-00:02:31.847188 & SMB Name Wildcard & 200.216.167.81:1025 & 130.85.133.252:137
Pretty simple, right? Now I could used AWK with the "&" as a delimiter to print just the IP:Port pairs and pipe that into a new file. For instance:
cat alertinternal.txt | awk -F"&" {print $3" "$4}
This will give me this kind of a file:
200.216.167.81:1025 & 130.85.133.252:137

I next replaced all the '&' symbols with ":". Now, I can use ":" as a delimiter to for all the different combinations I want, like:
cat alertinternal.txt | awk -F":" {print $1" "2"} gives me the source IP:port pairs, and so on...
Scan\OOS Files: I basically did all the same things I did with the alert files, but instead of just breaking up the files into external and internal scans I also broke them out into type. I greped out all instances of "UDP', and "SYN", and what was left over were grouped under a file called 'otherscans' (those TCP scans with flags other than the SYN set). The only real difference is how I delimitated the scan files at first. I used this ugly command to chop off the timestamp entries in both all the files:
cat scaninternalTCP.txt | cut -c 17- > scaninternalTCP.txt
By looking at the files the knew the timestamp was 16 characters in length so I issued the cut command to chop off 17 characters from the beginning to give me the lines starting with the IP's. I also piped the TCP files and Other files into Snort_Snarf but that was of little use to me in the overall conclusions. Because the UDP scans were too large to use Snort_Snarf I did use a program called WinSnort2HTML to give me a pretty good summary of what was going on, and it can be configured to sort by data, IP, port, and so on. It also provides links to the alerts and port information.
Calculations: To sum up all the data pairs I was interested in, I would utilize the sort and uniq -c commands. For instance, this will tally up how many times an IP went to a separate IP.
awk -F":" {print $1" "$3} alertinternal.txt | sort | uniq -c | sort -nr > IPtoIP.txt

That's basically how I managed all the data for my methodology.

Correlations From Previous Practical Submissions

Here is the list of the papers, *or other sources*, that best correlate with my detects and approach throughout my analysis.


Correlation.rtf

APPENDIX

Assignment Two: Detect # 1 Snort Alerts and Packet Dumps


Detect1Alerts.txt


Detect1Dumps.txt

Assignment Two: Detect # 2 Snort Alerts and Packet Dumps


Detect2Alerts.txt


Detect2Dumps.txt

Assignment Two: Detect # 2 PropFind DoS Exploit


Detect2Exploit.txt

Assignment Two: Detect # 2 Srcgrab Script


srcgrapExploit.txt

Assignment Two: Detect # 3 Snort Alerts and Packet Dumps


Detect3Dumps.txt


Detect3Alerts.txt

Assignment Two: Detect # 3 Traversal Scripts


Phusion.txt


Phusionexp.txt

Assignment Three: Kazaa Scripts


source.txt


Kazaa GET.txt

Assignment Three: Process_Scanalert Perl Program


ssp3.pl

References

"Online Resources.rtf"