Global Information Assurance Certification Paper

Intrusion Analysis

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 3.3

Saro Hayan

June 27, 2003

INDEX

STATE OF INTRUSION DETECTION

Intrusion Prevention - Just a Buzzword?

The security threats that face the networks of today have grown in complexity and threat level. Network administrators now must deal with attacks that vary from simple to crippling, and it is often very difficult to differentiate between the two at onset. Private companies, government agencies, educational institutions, etc. now rely heavily on the data network for their day-to-day operations. Most of these institutions invariably have external connections. These external connections could consist of peering connections, connections to private networks, connections to the Internet, etc. These networks must be protected from intrusion from outside entities and from internal attacks in some cases.

Network administrators must be able to protect the enterprise while still taking into consideration mission critical services. The first step in security is almost always a firewall to protect the network at the perimeter or to protect mission critical resources. Some companies have several firewalls giving them layers of security and segmentation in order to mitigate the damage in case systems are compromised. The inherent problem with firewalls is that certain traffic must be permitted through for business functionality. Most firewalls will have rule sets that permit or deny traffic based on Layer 3 and Layer 4 (OSI model) information. Certain types of firewalls do inspect a bit deeper into the traffic, but usually at a cost of becoming a bottleneck.

Before we go on, we should quickly look at the types of firewalls and the problems they leave administrator. There are several types of firewalls (or filtering devices, as routers could play this role in some instances), below is a short explanation of some of the most often used types of firewalls, and the some pros and cons of each.

Static packet filter - This is a device that has a static rule set inbound and/or outbound. It usually permits or denies traffic based on various criteria, such as a source/destination address, source/destination port and protocol. The device and rule set have no real intelligence; they do as they are configured. A good example of this is an access control list on a router to permit HTTP traffic to a web server. The filtering device has no idea what the payload of the traffic it is permitting. It would permit traffic with malicious intent along with normal traffic.

> Major Pros: They are very fast and not very resource intensive.
> Major Cons: They, inherently, must permit traffic, but have no way to know the content and intent of that traffic. They are very easily bypassed by a variety of methods.

Stateful packet filter – These types of firewalls have some intelligence, as there usually is a trust level associated with each interface, and have a concept of "state". An example of this would be where the traffic initiated from the "trusted" interface (for example, a company employee) would be permitted through to the "un-trusted" interface (for example, connection to the Internet). The firewall would build a state table entry recording the particulars of the session being started. The corresponding return traffic would be dynamically permitted back through.

Major Pros: These are also very fast and provide additional security by keeping "state" information. Certain traffic is permitted dynamically as desired, while others of that same type of connection are dropped as desired.
Major Cons: As with static filters, these devices usually make their decisions based on Layer 3 and Layer 4 information. They do not, as with static filters, have any knowledge of content and intent at higher layers.

Statefule inspection – These take the stateful packet filter to the next level by having the ability to look deeper into the payload (Layers 5-7) of certain types of connections and make decisions based on those criteria. FTP would be a very good example of this (PORT mode). The firewall would look deep enough into the packet to determine the TCP port negotiated for the FTP data channel. It will dynamically be permitted through.

Major Pros: Provide the additional ability to look into the payload of some types traffic. They are still rather fast.
Major Cons: Do not look at *all* the payload, only for specific types of traffic.

Proxy firewalls – These firewalls function in a much more intrusive manner. Client connections are directed at the proxy firewall where they are checked for validity based on some rule-set, then a connection on behalf of the client (hence it's called a proxy since it acts as one) is made by the proxy. The return traffic is forwarded on to the original client once received by the proxy. These types of firewalls are able to look deep into the payload (Layer 5-7) of every connection, and the rules can be made based on payload. These can be considered to perform functions similar to what we might look for out of Intrusion Prevention Systems, more on this later.

Major Pros: Filtering decisions can be made based on up to Layer 7 information as they look at all traffic payload.
Major Cons: Speed and throughput are compromised. They are very resource intensive and are often the bottleneck of a network. Many types of application will simply not work through these types of devices.

Most networks have at least one of the above types of devices (or some variation of the above types), some will have more than one in more of a layered design. These devices are usually deployed at the borders and provide a "crunchy outer shell" of sorts. This usually leaves a "soft interior" in most cases. Also, unless an institution is utilizing a proxy type device, there is traffic that is permitted through the perimeter without any knowledge of the content. For these reasons the use of Intrusion Detection Systems (IDS) became more prevalent. The two popular implementations of IDS are the host bases IDS (HIDS) and the network based IDS (NIDS).

The IDS has taken on the role of the "other" layer of security. If the firewalls inherently have to let certain traffic pass, the network administrator needs to identify the malicious portions of that permitted traffic. The IDS would be deployed at critical points of the network to detect any sort of malicious activity. The key here is the word DETECT. The intention of the IDS, in the purest sense, is to detect intrusions and alert the administrator(s) who are hopefully paying attention. There are some limited prevention capabilities implemented in some types of IDS that we will discuss in more detail below.

As mentioned earlier, there are two popular implementations, NIDS and HIDS. We will look at each bit more closely and identify their pros and cons.

The NIDS sits on the wire of a network segments and watches all the traffic on that segment. With the advent of switched networks, the placement of the NIDS is one of the biggest concerns. The NIDS must be placed in such a way that it can "see" all of the traffic that it is being asked to watch. Most enterprises utilize mirror ports from their switches or physical taps to ensure the NIDS sees all the traffic. The NIDS, once it identifies malicious traffic, can take action in a several ways. Some examples are that it can reset a TCP connection, or it can automatically log into a router or a firewall and add a rule blocking a particular host.

Major Pros: NIDS can identify malicious activity for a variety of Operating Systems (this is a pro and a con). The NIDS usually has less chance of being tampered with itself. It also provides a big picture view into the segment it is watching.

Major Cons: The NIDS is not OS specific. It can invariably miss certain OS specific traffic. There can be many false positives as well with a NIDS. Because it is not the intended target of a communication, it must re-assemble all the streams that it sees. It may not do this correctly in given situations. It cannot deal with encrypted traffic. If the segment is flooded or experiences high volume; it can drop packets or not see everything that is occurring. The cons of IDS will be discussed more later.

The HIDS is essentially software that runs on a host computer, a workstation or a server, that is specific for that operating system; possibly even for the software packages that are installed on that particular host. It looks at malicious behavior that is aimed directly at that particular host. It can take action in various ways at the operating system level or at the application level. The HIDS implementations could more easily provide some intrusion prevention.

Major Pros: They are OS and/or application specific and therefore can provide highly detailed information with regards to malicious activity. They can also respond with highly detailed and specific responses.

Major Cons: HIDS can be expensive and difficult to deploy on a large enterprise. They can also be difficult to manage. Centralized management and reporting have been improved, but with thousands of hosts to deploy on, it is still a rather hefty task. System performance is often degraded due to the HIDS. Since the system in question is the target, an intruder can often corrupt the log data if the system is indeed compromised.

A NIDS or a HIDS use certain techniques to identify malicious traffic. What makes a certain traffic pattern valid or malicious is determined by the detection mechanism and rules in place for that mechanism in the NIDS or HIDS. There are a variety of different types of detection, below are some of the common implementations.

Signature based – The IDS looks for specific patterns in traffic that are known to be of malicious intent. It can use stateful detection or it can detect malicious activity on a packet-by-packet basis.

Major Pros: The signatures can be very specific. The signature can look for a specific part of session to determine if the flow is malicious.

Major Cons: All the detection is based on a database of signatures of well-known attacks that are static. It is reactionary by nature, an attack is identified, a signature is created,

and that signature is implemented. The attacks always precede the signatures and are often too rapid and consequently the signatures are not updated fast enough on the enterprise. Also, developers must continuously keep up with updated signatures. Possibly most important, another negative of signature based IDS are false positives. The IDS will on occasion identify normal traffic as being malicious. They will also on occasion miss identifying malicious traffic.

Anomaly/Behavior based – The IDS looks for transport or protocol anomalies. There are some types of traffic that are never valid; these types of anomalies are detected. Also, the network traffic is base-lined and the IDS is used to detect any variations from that baseline. There are no predefined signatures to rely on for these types of activities..

Major Pros: This type of IDS can identify attacks without the need of specific signatures. It can detect new attacks in addition to well-known ones. Also, less updates are required from developers because they don't have nearly the same dependence on signatures. The number of false positives is usually much lower with this type of IDS.

Major Cons: There is a lack of information from the attack since the behavior rules are not based on a specific attack. An attack that doesn't look anomalous would be able to go by undetected.

Anomaly and Signature (Hybrid) based – This IDS combines the best of the signature based IDS and the best of the anomaly based IDS. This gives the administrator the best of both worlds, but it also carries with it the shortcomings of both worlds. The pros and cons of this type are similar to that of the signature based and the anomaly based IDS. One type does not make up for the deficiencies of the other. They inherit the problems of one another.

IDS is not plug and play. Most are also built on the thought that they would detect the attack and send an alert. The administrators must tune the IDS so they do not drown in alarms. This is, to put it bluntly, a never-ending pain in the neck. IDS management is not easy work. False positives have been one of the biggest drawbacks to IDS alerting. Administrators must constantly tune the IDS so that the information that it reports is valid. This usually requires knowledge of network resources, operating systems, vulnerabilities, etc. When an IDS alarm is received by administrators, some of the things they must think about are: What is the attack (new/old)? Does it affect my systems? Is the vulnerability prevalent on my network? Did it succeed or fail? Is it ongoing? Implement some countermeasures if it is ongoing, such as blocking that host with a rule in the firewall. These are just some of the things that an IDS admin has to think about at the onset of an alarm.

To add to the complexity, administrators will have to deal with a high number of false positives. This can cause the administrator to be less meticulous when dealing with the alarms, which can result in real attacks slipping through. In a pure sense, the IDS should take no action on the attack; it should only alert the administrator of it. Some IDS have added such features as the ability to manipulate firewall or router rules or to send TCP resets to stop a malicious connection. These automated features can work fairly well if they are implemented properly. But, if an attacker finds a weakness in the implementation or if a new vulnerability gives the attacker the upper hand, they can cause the IDS to denial of service the very network it is trying to protect. They need to be implemented very carefully.

Firewalls work well but must permit through some traffic without knowing the content. IDS can be a useful tool in detecting intrusions, but require a great deal of time and effort to be made efficient. False positives are no fun at all and seem to be never ending. So what next? With all of that said, are we ready to put the IDS inline…? Maybe!

Security technologies mature and evolve. Things that were once buzzwords are now realities. There are many types of firewalls and Intrusion Detection Systems in addition to a large number of other security products. IDS has not developed completely. We went through many of the negatives about IDS above, but there are things that have come a long way. For example, gigabit capturing capability is a reality. Companies have put quite a bit of effort in developing specialized processors that provide speeds that were not available just a few years ago. Also, many types of the anomaly detection are done with very little false positives. Specific functions of the IDS are ready for the "big time".

The philosophy behind Intrusion Prevention Systems (IPS) is to somehow combine the all positives of firewalls, the positives of IDS, add some new capabilities and put them into one device. It would be an inline device that utilizes the capabilities of a firewall and an IDS to make decision on the traffic passing through it. That sounds easy enough. Assuming that all the hardware parts were in place, the device should be able to have firewall type rules, but would also be able to look at the payload of the traffic it processes. It would actively make decisions of what to do with the traffic based on Layer 2 through Layer 7 of the OSI model. It could have the ability to perform virus detection since it is looking at the payload inline. False positives would have to be a thing of the past since the device lives in-line. If an IPS had as many false positives as many of today's IDS have it would be a complete failure. With IDS, the administrators make the final decision in most cases. In some instances, the limited active response available on IDS is used. In the IPS world, it would make the final decision on what should be done with the traffic based on the rules it is configured with. It could have a variety of responses such as permit, deny, redirect, rate limit, deny with a TCP RESET, etc. The IPS would have to do all or most of these things. To top it off, it would have to do all of these while avoiding becoming the bottleneck in the network.

You can take an IDS and configure it to be "inline", that's simple. The question that needs answering is; how many of the rules would you feel comfortable activating? How many of the rules in the IDS do you think have a low enough false positive rate that you would feel comfortable always dropping traffic that matched those signatures? These are the questions that need to be answered when jumping into the IPS world. When the IDS makes mistakes, you get more logs to muddle through. If the IPS makes mistakes, you may get pink slips to muddle through.

I believe that the technology is nearly there for a successful implementation of an IPS. The first generation devices will probably have the very basic and concrete rule sets. You would probably add to that a stateful inspection firewall function. Those two components combined would be a tremendous gain in security. The devices would now be able to make more and more decisions at higher layer protocols. The goal of the IPS would be to take the best features of firewalls, the best features of IDS, mix them all together in a big blender, and come out of it with a new

device. In my opinion, Intrusion Prevention will become a normal part of the network; whether it is the appliance protecting the enterprise, or the host based IPS protecting your particular Operating System from attacks.

NETWORK DETECTS

DETECT 1

The usage and specifics about commands is at the end of the analysis in the NOTES section. References to the NOTES sections will be noted with [SECTION_NAME].

1. Source of Trace:
The trace used for this analysis was from:
http://www.incidents.org/logs/Raw/2002.10.15

The traces contain only the packets that violate the rule set and have been sanitized. More detail regarding the traces can be found here:
http://www.incidents.org/logs/Raw/README

To help in the analysis of the chosen trace, I did some prep work before getting started. I downloaded all the traces from www.incidents.org/logs/Raw dated October of 2002 (10/1 - 10/18). I then used a tool called "pcapmerge" to merge the files together. This was done to help with the analysis by giving an overall picture. The process created a single file called "all-oct" for the data from October. [PCAPMERGE]

Next I created the Snort alert file for the particular trace that I chose (2002.10.15). Given the tcpdump binary format trace files, I used Snort version 2.0.0 (Build 72) with a current rule set and all rules enabled. The following command was used [SNORT]:

gcia# snort -r 2002.10.15 -c /usr/local/snort/snort.conf -l ./10-15

This produced an alert file for the specific day we are concerned with. Below is a sample record out of that alert file:

[**] [1:524:6] BAD-TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/15-05:34:50.446507 211.47.255.24:41104 -> 170.129.195.40:0
TCP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq: 0xD30F0032 Ack: 0x0 Win: 0x16D0 TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0

To help analyze the alert files, I used a tool called "Snortsnarf" (http://www.silicondefense.com/software/snortsnarf/index.htm). Snortsnarf is a Perl tool that takes Snort alert files as input and produces easy to read and easy to analyze HTML reports (for lack of a better word). Obviously you must have Perl installed and it is recommended that the

Perl time modules written by David Muir Sharnoff (Time::JulianDay) be installed first. The following command is used to accomplish the task at hand [SNORTSNARF]:

gcia# snortsnarf.pl -d ./snortsnarf10-15 ./10-15/alert

This step helped show that there were 44 alerts with 2 source IP addresses and 3 internal destination IP addresses. The sources were 211.47.255.23 and 211.47.255.24. The 3 internal destinations were 170.129.21.249, 170.129.195.40 and 170.129.23.96.

Before moving on, I created a tcpdump binary file for all of October for later use. The file only included the packets that we are interested in (TCP port 0). Tcpdump version 3.7.2 was used to accomplish this [TCPDUMP].

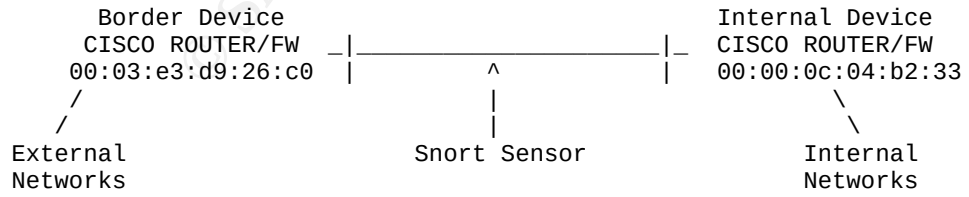
gcia# tcpdump -nve -r all-oct tcp port 0 -w all-oct-tcp0

These steps produced the original trace file (2002.10.15), a file (all-oct) that contained all the traces from the entire month of October and a file (all-oct-tcp0) that contained only packets destined for TCP port 0 from the month of October. Next on the agenda was getting the "lay of the land", so to speak.

I manipulated the trace file a bit (see MAC address section in the Notes section below). I discovered that there were only two MAC addresses in the 2002.10.15 trace that were the source or destination of every frame. To verify this, I looked at the source/destination MAC addresses from the entire month. They were all the same two addresses we found in our original trace. These addresses were [MAC_ADDRESS]:

00:00:0c:04:b2:33 and 00:03:e3:d9:26:c0.

Both addresses are registered as Cisco Systems devices (see http://standards.ieee.org/regauth/oui/oui.txt). These two MAC addresses are the only ones communicating on this segment. Some of the types of devices that would fit this mold would be routers, firewalls and proxy devices. Cisco only has a web proxy (cache) product but there is much more than just web traffic on this segment from these devices which led me to eliminate the possibility either of these devices were proxy device. The following is the network layout that would make the most sense considering the above information.



The sensor can be implemented in a variety of ways. It could be utilizing a mirror/span port off of a switch between the two devices. One of these devices may have the ability to have a mirror/span port configured and the sensor could reside there. There could be a physical tap on the line between the two devices. Also, it is possible that there is a hub between the two and the

sensor is sitting on one of the ports of the hub. There are other implementation methods that I haven't mentioned of, but it would be safe to say most IDS implementations fit one of these.

One of the last things I wanted from this segment was information about the rule set implemented on these devices. This could help determine the type of device (router/firewall/stateful/static filter/etc...). There isn't enough information in the traces to make statements regarding the rules with any certainty, but there is enough information to get a good idea.

A simple way to determine this is to look at connection initiation attempts (packets with SYN bit set) that are destined for our internal networks. This is done by telling tcpdump to look at the 13th byte offset of TCP header (the flag byte of the TCP header) and making sure its value equals what we are looking for. In this case I would like the TCP flag byte value to equal 2, which indicates that the SYN bit and only the SYN bit is set (reference Snort 2.0.0 User's Manual for detail on the TCP flag byte). NOTE: After analyzing our trace, 170.129.0.0/16 seems to fit for the internal network).

The following commands will accomplish this:

gcia# tcpdump -n -r 2002.10.15 'tcp[13] = 2 and dst net 170.129.0.0/16' | awk '{print $4}' | awk -F. '{print $5}' | sort | uniq

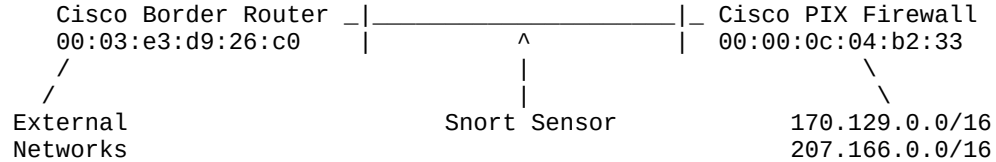
The additional awk, sort and uniq commands help us get a list of the destination ports that are being permitted through border router (there are probably many others, but these are definitely there according to our evidence). The ports are TCP 0, 53, 1080, 3128, and 8080.

We can also look at UDP traffic destined for UDP ports on the inside with the following command:

gcia# tcpdump -n -r 2002.10.15 udp and dst net 170.129.0.0/16

The handful of lines that are filtered out with these commands show that there is traffic destined for UDP port 0 and 111 on the inside. The Neohapsis listing of ports was used to get more information about the ports we found (http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html)

There is a variety of source and destination IP addresses in the above packets. This likely indicates that they are being permitted through border without any real control. I would guess that the border device is a static filter at best, likely a router. It is difficult to make many more assumptions from the given data since they aren't full captures. From the information we have gathered, the revised layout would look as follows:



2. Detect was generated by:
Snort version 2.0.0 (Build 72)
Snort signature 524 generated these alerts. More information can be found here:
http://www.snort.org/snort-db/sid.html?id=524
The exact signature was:

alert tcp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD-TRAFFIC tcp port 0 traffic"; classtype:misc-activity; sid:524; rev:6;)
For this signature the Snort rule action is "alert", identified by the first word of the rule. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 actions available by default, alert, log, pass, activate, and dynamic (for more information on these refer to the Snort 2.0.0 User's Manual).

This rule is for bidirectional (identified by the <> bidirectional operator) TCP traffic where port 0 is used by the internal network(s) as defined by the variable $HOME_NET. That is one of two variables that we see in this signature, the second is $EXTERNAL_NET. Both are variables that are usually defined in the snort.conf file to indicate which networks are internal and which are not. I don't want to go into the second part of the rule, refer to the Snort manual for more detail.

3. Probability the source address was spoofed:
I used some passive fingerprinting techniques on the packets to help with this issue. I used the host identifier table from a passive OS fingerprinting tool called p0f. Below is sample packet from the trace:

13:08:37.996507 211.47.255.23.46919 > 170.129.23.96.0: S [tcp sum ok] 2202284014:2202284014(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 46, id 0, len 52)

The first thing to look at is the window size. All the interesting (destined for TCP port 0) packets have a window size of 5840. Looking at p0f (available at http://www.stearns.org/p0f/p0f.fp), we can see that there are eight possible OS choices:

Win	TTL	MSS	DF	WS	sOK	nop	LEN	OS Description
5840	64	1460	1	0	1	1	60	Linux 2.4.2 - 2.4.14 (1)
5840	64	1460	1	0	1	1	52	Linux 2.4.1-14 (1)
5840	64	1460	1	0	1	1	48	Linux 2.4.1-14 (2)
5840	64	1460	0	0	1	1	60	Linux (unknown?) (2)
5840	64	1460	1	0	0	1	60	Linux 2.4.13-ac7
5840	64	1460	1	223	1	1	60	Linux-2.4.13-ac7
5840	128	536	1	0	1	1	48	Windows 95 (3)
5840	128	1460	1	-1	1	1	48	Windows 95 or early NT4
5840	64	1460	1	0	1	1	52	-Source IP of our trace-

Using the table above (a snip directly from the p0f.fp file), we can quickly rule out the Windows 95 systems due to TTL. We can then rule out the "Linux (unknown?)" OS because of the lack of DF bit being set. The packet size (LEN) field will then eliminate all the other Linux types to leave us with a Linux 2.4.1-14 (1) kernel OS.

A quick whois query of ARIN (www.arin.net) identified the source network (211.47.255.x)as part of a block of addresses that are part of APNIC (Asia Pacific Network Information Centre). A quick APNIC whois query (www.apnic.net) gave the following:

```
inetnum:   211.47.255.0 - 211.47.255.255
netname:   ORG84651-KR
descr:     SAEROUNNET
descr:     789-28 sihungdong kumchungu
descr:     SEOUL
descr:     153-034
country:   KR
admin-c:   CK724-KR
tech-c:    IJ161-KR
remarks:   This IP address space has been allocated to KRNIC.
.../snip/...
```

Along with these ARIN and APNIC queries, we did a search of the DShield (www.dshield.org), myNetWatchman (www.mynetwatchman.com), and DeepSight Analyzer by SecurityFocus (http://analyzer.securityfocus.com) databases for incidents involving the source IP addresses. Dshield's database indicates that the last complaint regarding these IP addresses was sent in April (it varies for each of the 5 IP addresses). myNetWatchman also indicates a number of incidents with regards to these IP addresses and as with DShield, the last noted ones were in April. DeepSight also indicates the latest incident in March. NOTE: DeepSight Analyzer is a subscription type of service.

The results of these queries got me interested in whether this network was still active or not. After a few basic connectivity tests that failed, I decided to check some internet BGP tables to see if this route is currently being routed on the internet. (www.traceroute.org has several such route servers/proxies)

```
gcia# telnet route-server.ip.att.net
#################### route-server.ip.att.net ####################

route-server>show ip bgp 211.47.255.0
% Network not in table
```

The above command would display any BGP advertisement that included this particular prefix. According to this output, there are none. I checked several other route servers/proxies, but the results were the same. Was this net ever routed on the internet? I looked back at some historical BGP data (http://archive.routeviews.org). This network was still being advertised in March of 2003 as part of a larger block. From data dated March 15th, the following was being advertised:

```
* 211.47.224.0/19 217.75.96.60             0 16150 8434 2119 8210 4637 3976 9487 i
```

The things of note in this output are the network advertisement (211.47.224.0/19) and the final AS (Autonomous System) in the AS path (9487). This network block encompasses the addresses we are concerned with and the last AS on the list is the AS which advertised it. So according the internet routing tables, this network stopped being advertised or routed sometime after March.

Back to the question of whether the source addresses were spoofed or not. I do not believe these were spoofed sources. The source network and AS are/were legitimate (not that that means much

by itself). There have been many complaints filed against addresses from this block and specifically these addresses. Also, the fact that these are lone SYN packets would support the assumption that these are not spoofed. The SYN would elicit a response but the real source of the packets would never see them if these addresses were spoofed.

4. Description of attack:

At first glance, this looks like regular session establishment attempts (except for the TCP port 0 part). But upon further review, I believe this to be some sort of OS fingerprinting attempt. The probing device is attempting to use the responses received to his port 0 probes to accomplish this. The packets look to be crafted, but there is more here than meets the eye. These do not look like crafted packets sent by Nmap, hping, Nemesis, Rafale, Packit or a few other packet automated crafting/scanning tools that I've used. The intervals between packets is 3, 6, 12 seconds. This is typical for a TCP connection retry timer. It is possible that these packets were crafted and then sent on their way in a fashion more consistent with a normal Linux OS IP stack. There are many packet crafting scripts out there and beyond this, it is possible that this is a "home brew" script. The fact that the IP ID never increments on each retry leads me to believe that these were crafted packets (good read about IP ID and Linux 2.4x kernel in this post by Ofir Arkin, http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1). An IP ID of 0 is not illegal, but normal TCP traffic would increment the IP ID on every retry. I've tried this with several OSes and found it to be the case.

To summarize, these seem to be crafted packets because the IP ID does not increment on retry attempts. They seem to be using some normal Linux operating system function to be sent to the target hosts from a real IP address. I would guess that this is some sort of OS fingerprinting attempt, in other words reconnaissance. I would assume this is either being done with a tool that I am not familiar with, options with current tools that I am familiar with, or some combination of packet crafting and using normal Linux OS functions to send the packet.

5. Attack mechanism:

The probe is sending SYN packets with non-incrementing IP ID to hosts on TCP port 0. Otherwise the traffic pattern is that of a normal connection attempt. There is no well known service on TCP port 0 nor is there any known vulnerabilities to such probes. I believe it network reconnaissance and information gathering attempt to try to determine the OS of the destination by the response received.

6. Correlations:

Much of this was done while trying to determine if the IP address was spoofed in the section above. I did not find any specific information on the use of TCP port 0. I also did not find any specifics positive or negative results of such a scan. This network went "dead" between now and March sometime as noted by the disappearance of its routes from the internet BGP routing table. Most of the incidents reported by DShield, myNetWatchman, and DeepSight Analyzer were of web (IIS mostly) type attacks. Unfortunately most of the incidents did not have very much detail, possibly because they were not very current.

7. Evidence of active targeting:

If I was asked to give a yes or no answer, I would lean towards no. There is no evidence that the probes are directed at active hosts. This could be just a random scan. But I believe it is difficult to make any claim without more information regarding the destination IP addresses. Are they active addresses? Are they critical systems? Is there a common application/service? etc. The destination IP addresses do look to be rather random, but without details about the internal network I would not disregard the possibility of active targeting.

8. Severity:

The formula:
severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality - We do not have information on the targeted IP addresses. Some testing I've done indicates that most current OSes will simply treat it as any other connection attempt to a closed port responding with a RST-ACK packet. I gave this item a 3 because of the unknown factor.
Lethality - If the assumption that this is simple information gathering is correct, this number would be lower. But, since we do not know with any amount of certainty, I would give this item a 3.
System countermeasures - We have no information about the internal systems that were targeted. We do not know if this was blocked or if the hosts simply did not exist. This item gets a 2 because of the unknown factor again.
Network countermeasures - The traffic was either blocked or the hosts did not exist (both can result in retries). This item gets a 2 as it doesn't look to be getting to any systems but it is getting past the border.
The answer to the formula:
Severity = (3 + 3) - (2 + 2)
Severity = 2

9. Defensive recommendation:

The attack was likely blocked but did get past the border router. The hesitation is because the IP addresses being targeted may not have been live. The trace would look the same in either case. I would recommend this traffic be blocked at the border. It is something should never be permitted to enter or leave our network. There is no known service or use for it. On a Cisco router the following could easily block any communication on TCP port 0 if applied (or added to) the ingress and egress access control lists:

access-list 101 deny tcp any any eq 0
access-list 101 deny tcp any eq 0 any

10. Multiple choice test question:

What would make these packets a normal TCP retry attempt?

```
05:34:50.446507 211.47.255.24.41104 > 170.129.195.40.0: S [tcp sum ok]
3540975666:3540975666(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 46,
id 0, len 52)
05:34:53.296507 211.47.255.24.41104 > 170.129.195.40.0: S [tcp sum ok]
3540975666:3540975666(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 46,
id 0, len 52)
```

```
05:34:59.466507 211.47.255.24.41104 > 170.129.195.40.0: S [tcp sum ok]
3540975666:3540975666(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 46,
id 0, len 52)
05:35:11.276507 211.47.255.24.41104 > 170.129.195.40.0: S [tcp sum ok]
3540975666:3540975666(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 46,
id 0, len 52)
```

A) Positive integer value for TCP destination port
B) Positive integer value for IP ID field
C) 1 second interval between retries
D) Incrementing of the IP ID
E) This is a normal connection initiation attempt.
Answer: D

Detect 1 - NOTES

PCAPMERGE
pcapmerge -r 2002.10.1 -r 2002.10.2 .../snip/... -r 2002.10.18 -w all-oct
-r - input file(s)
-w - output file

SNORT
Snort command line options used:
-l - Log to a specific directory
- The directories must already be created before running this command
-r - Read and process tcpdump file
-c - Specify location of rules file

SNORTSNARF
Snortsnarf command line options used:
-d - Sets the output directory for the HTML files that are created
The syntax is very simple, snortsnarf [options] [alert file]. By default it outputs to the working directory.

From the Snortsnarf pages created for my alert, I gathered the following important bits of information:

```
Priority | Signature (click for sig info) | # Alerts | # Sources | # Dests |
Detail
---------|--------------------------------|----------|-----------|---------|-
--------
3        | BAD-TRAFFIC tcp port 0 traffic | 44       | 2         | 3       |
Summary
.../snip/...
```

And also more detailed by IP address:

```
Sources triggering this attack signature
Source        | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts
(total)
--------------|----------------|------------------|--------------|-----------
-----
211.47.255.24 | 30             | 30               | 2            | 2
211.47.255.23 | 14             | 14               | 1            | 1
```

```
Destinations receiving this attack signature
Destinations   | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs
(total)
---------------|----------------|------------------|--------------|----------
-----
170.129.21.249 | 15             | 15               | 1            | 1
170.129.195.40 | 15             | 15               | 1            | 1
170.129.23.96  | 14             | 14               | 1            | 1
```

TCPDUMP

Tcpdump command line options used:

-w - Write the raw packets to file
-n - Do not do name lookup on IP addresses in file
  - We do this to make things go much faster since DNS lookups don't have to happen
-v - Verbose, gives more information such as the TTL, IP ID, LEN (Packet Length) and checksum information
  - The additional information will be key in determining whether the packets were spoofed or not (more later)
-e - Displays link layer header information, which includes source and destination MAC address
  - We would like to determine the Layer2 topology, using that information will possibly give us the layout of the network

The expression used to filter out all but the needed results was 'tcp port 0'. In this expression, the first field identifies the protocol we are looking for (TCP), the second and third fields identify the port in question (0).

MAC_ADDRESS

```
giac# tcpdump -nve -r 2002.10.15 | awk '{print $2, $3}' | sort | uniq
0:0:c:4:b2:33 0:3:e3:d9:26:c0
0:3:e3:d9:26:c0 0:0:c:4:b2:33
```

For this step, I used the unix text manipulation tools of "sort", "uniq" and "awk" to help with the analysis. I found that through the entire trace file, there were only two source and destination MAC addresses. To support this information, I ran the "all-oct" file created above from the entire month's worth of traces through the same command with the same results:

```
gcia# tcpdump -nve -r all-oct | awk '{print $2,$3}' | sort | uniq
0:0:c:4:b2:33 0:3:e3:d9:26:c0
0:3:e3:d9:26:c0 0:0:c:4:b2:33
```

Intrusions.org Mailing list Post – posted 6-24-2003

http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00338.html

One recommendation I'd like to make is to read this (and perhaps your entire practical) as a reviewer would. For example, use more "I" or "you – meaning the reader" rather than "we". When I read some paragraphs, it almost infers that you worked on this as a member of a team - of course you did not *grin*.
From: "Dave van Nierop" <dvn@campana.com>

The fact that the IPID never increments on each retry leads me to believe that these were crafted packets (good read about IP ID and Linux 2.4x kernel in this post by Ofir Arkin, http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8 &oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1). An IP ID of 0 is not illegal, but normal TCP traffic would increment the IP ID on every retry. I've tried this with several OSes and found it to be the case. Ref: http://www.sys-security.com/archive/bugtraq/ofirarkin2002-02.txt says: The Linux Kernel 2.4.x way: Linux Kernel 2.4.x is using IP ID values of zero in several circumstances, whenever the DF is set: The IP ID = 0 do not really indicate that it is a crafted packet.. Some kernels do have ID = 0 when the packet is not fragmented and DF = 0 What do you think ?
From: Ashley Thomas <athomas@cc.gatech.edu>

---------response-------
Yes, I agree with you completely. IP ID of 0 by no means implies that it is a crafted packet. There were a few factors as to why I thought the packet was crafted. The trace that I had contained all SYNs. Ofir specifically talked about SYN/ACKs with the DF bit set having the possibility of an IP ID of 0. In fact, I did some simple tests and saw just that. The problem I had with the this trace was that they were all SYNs (I haven't seen non-changing IP ID of 0 anywhere with regards to SYNs) and that each retry attempt, the IP ID did NOT increment. That's why I thought the packets were crafted. Again, thanks for taking the time.
Saro

Hey Saro,
Great Job! In looking through it the only things I felt I could comment on where kind of petty however I will shot them off anyway.
##
Comment 1
> The SYN would elicit a
> response but the real source of the packets would never see them if these
> addresses were spoofed.

If an attacker has control of a router that is located in the return
path the attacker could use a spoofed address to elicit a response.

---------response-------

THanks for commenting...
> Comment 1
> The SYN would elicit a
> response but the real source of the packets would never see them if these addresses were spoofed.
> If an attacker has control of a router that is located in the return
> path the attacker could use a spoofed address to elicit a response.

Couldn't argue that point much, good thinking.

DETECT 2

1. Source of Trace:
This trace was from my work network.
The alerts were collected from Sourcefire (http://www.sourcefire.com/) sensors. Also, tcpdump was used to capture the questionable activity as it was happening. The trace and set of packets we were interested in were a large number of fragments that had the DF (don't fragment) and the MF (more fragments) bits set. That would seem to be anomalous at first glance. I wanted to investigate.

The following is the sample screenshot of these particular events.

Event Data

Message	BAD-TRAFFIC bad frag bits	Generator ID	1
Classification	Misc activity	Snort ID	1322
Priority	3	Revision	5
Rule	alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC bad frag bits"; fragbits:MD; sid:1322; classtype:misc-activity; rev:5;)		Deactivate

Ethernet Header

SRC MAC:	00:D0:05:40:13:FC	DST MAC:	08:00:20:96:1E:62	Type:	0x0800

IP Header

Version:	4	Header Len:	5	TOS	0	Total Len (in bytes)	1500
16-bit ID	41933	Frag Flags	DF MF	13-bit offset		0x0000 (0)	
TTL	62			Protocol		UDP	
Source IP				X.X.176.144			
Destination IP				Y.Y.252.47			

After being notified of this alert, we were able to get a tcpdump of this exact activity and were able to capture the full conversation between these two hosts. We also used "snoop" on a Solaris system to capture the same traffic. Snoop is a packet capture implementation on Solaris. We found some interesting differences between the two, which I will get into later. The following commands were used for tcpdump and snoop respectively:

giac # tcpdump –i eth0 –s 1500 -X –w nfstcpdump.cap host Y.Y.252.47
giac # snoop –d hme0 –o snoop.cap host Y.Y.252.47

The tcpdump capture of an entire transaction is shown below (tcpdump –T rpc –nXv –r nfstcpdump). The "-T rpc" indicates there is rpc traffic and will force tcpdump to interpret them as such:

```
15:41:36.409929 X.X.176.144 > Y.Y.252.47: udp (frag 4680:928@7400) (ttl 63, len 948)
0x0000   4500 03b4 1248 439d 3f11 c42c XXXX XXXX        E....HC.?..,....
0x0010   YYYY YYYY 0000 0000 0000 0000 0000 0000        ................
.../snip/...
15:41:36.410356 X.X.176.144 > Y.Y.252.47: udp (frag 4680:1480@5920+) (ttl 63, len
1500)
0x0000   4500 05dc 1248 62e4 3f11 a2bd XXXX XXXX        E....Hb.?.......
0x0010   YYYY YYYY 0000 0000 0000 0000 0000 0000        ................
.../snip/...
15:41:36.410494 X.X.176.144 > Y.Y.252.47: udp (frag 4680:1480@4440+) (ttl 63, len
1500)
0x0000   4500 05dc 1248 622b 3f11 a376 XXXX XXXX        E....Hb+?..v....
0x0010   YYYY YYYY 0000 0000 0000 0000 0000 0000        ................
.../snip/...
15:41:36.410550 X.X.176.144 > Y.Y.252.47: udp (frag 4680:1480@2960+) (ttl 63, len
1500)
0x0000   4500 05dc 1248 6172 3f11 a42f XXXX XXXX        E....Har?../....
0x0010   YYYY YYYY 0000 0000 0000 0000 0000 0000        ................
.../snip/...
15:41:36.410660 X.X.176.144 > Y.Y.252.47: udp (frag 4680:1480@1480+) (ttl 63, len
1500)
0x0000   4500 05dc 1248 60b9 3f11 a4e8 XXXX XXXX        E....H`.?.......
0x0010   YYYY YYYY 0000 0000 0000 0000 0000 0000        ................
.../snip/...
15:41:36.411101 X.X.176.144.2049 > Y.Y.252.47.890970796: reply ok 1472 (frag
4680:1480@0+) (ttl 63, len 1500)
0x0000   4500 05dc 1248 6000 3f11 a5a1 XXXX XXXX        E....H`.?.......
0x0010   YYYY YYYY 0801 03fd 2088 b034 351b 22ac        ...........45.".
.../snip/...
15:41:36.411144 X.X.176.144.2049 > Y.Y.252.47.890970797: reply ok 112 (DF) (ttl 63, id
0, len 140)
0x0000   4500 008c 0000 4000 3f11 dd39 XXXX XXXX        E.....@.?..9....
0x0010   YYYY YYYY 0801 03ff 0078 a916 351b 22ad        .........x..5.".
.../snip/...
15:41:36.412752 Y.Y.252.47.0x351b22ae > X.X.176.144.0x6f: 168 set 100003.3 (DF) (ttl
255, id 30965, len 196)
0x0000   4500 00c4 78f5 4000 ff11 a40b YYYY YYYY        E...x.@.........
0x0010   XXXX XXXX 03ff 0801 00b0 c816 351b 22ae        ............5.".
.../snip/...
15:41:36.413134 X.X.176.144.2049 > Y.Y.252.47.890970798: reply ok 112 (DF) (ttl 63, id
0, len 140)
0x0000   4500 008c 0000 4000 3f11 dd39 XXXX XXXX        E.....@.?..9....
0x0010   YYYY YYYY 0801 03ff 0078 a915 351b 22ae        .........x..5.".
```

As can be seen here, without dumping the HEX (tcpdump –X option) of the packet, we would not notice the DF/MF bits being set. The highlighted fields above note the fragmentation bit settings. A hex 6 in this position indicates that both the DF and MF bits are set, a 4 indicates that the DF bit is set, and a 2 indicates the MF bit is set.

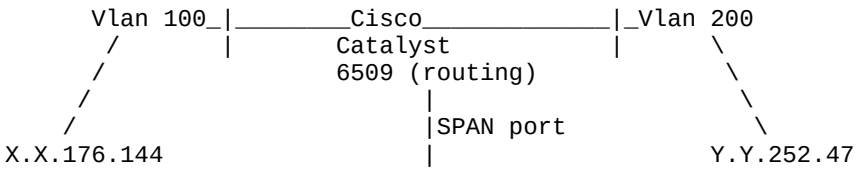
All the fragmented packets should have datagram length of 1500 (MTU of the Ethernet network it is on) except for the last fragment. The header length field for each can been seen highlighted above. 05dc HEX translates to 1500 and 03b4 translates to 948. This is normal fragmentation behavior.

A note of the these highlights in the above output. Tcpdump seems to think that these are the destination ports for these NFS (source UDP 2049) packets. This does not seem to be correct. Once I noticed this phenomena, I did the same capture with both snoop and tcpdump. Below is the output from snoop (snoop -V -i nfssnoop).

```
_____________________________________
1  0.00000 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 962 bytes
1  0.00000 X.X.176.144 -> Y.Y.252.47 UDP continuation ID=4680
_____________________________________
2  0.00042 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 1514 bytes
2   0.00042 X.X.176.144 -> Y.Y.252.47 UDP continuation ID=4680
_____________________________________
3   0.00013 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 1514 bytes
3   0.00013 X.X.176.144 -> Y.Y.252.47 UDP continuation ID=4680
_____________________________________
4   0.00005 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 1514 bytes
4   0.00005 X.X.176.144 -> Y.Y.252.47 UDP continuation ID=4680
_____________________________________
5   0.00011 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 1514 bytes
5   0.00011 X.X.176.144 -> Y.Y.252.47 UDP continuation ID=4680
_____________________________________
6   0.00044 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 1514 bytes
6   0.00044 X.X.176.144 -> Y.Y.252.47 IP  D=Y.Y.252.47 S=X.X.176.244LEN=1500, ID=4680
6   0.00044 X.X.176.144 -> Y.Y.252.47 UDP D=1021 S=2049 LEN=8328
6   0.00044 X.X.176.144 -> Y.Y.252.47 RPC R XID=890970796 Success
_____________________________________
7   0.00004 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 154 bytes
7   0.00004 X.X.176.144 -> Y.Y.252.47 IP  D=Y.Y.252.47 S=X.X.176.244LEN=140, ID=0
7   0.00004 X.X.176.144 -> Y.Y.252.47 UDP D=1023 S=2049 LEN=120
7   0.00004 X.X.176.144 -> Y.Y.252.47 RPC R XID=890970797 Success
_____________________________________
8   0.00160 Y.Y.252.47 -> X.X.176.144 ETHER Type=0800 (IP), size = 210 bytes
8   0.00160 Y.Y.252.47 -> X.X.176.144 IP  D=X.X.176.244S=Y.Y.252.47 LEN=196, ID=30965
8   0.00160 Y.Y.252.47 -> X.X.176.144 UDP D=2049 S=1023 LEN=176
8   0.00160 Y.Y.252.47 -> X.X.176.144 RPC C XID=890970798 PROG=100003 (NFS) VERS=3
PROC=1
8   0.00160 Y.Y.252.47 -> X.X.176.144 NFS C GETATTR3 FH=5054
_____________________________________
9   0.00038 X.X.176.144 -> Y.Y.252.47 ETHER Type=0800 (IP), size = 154 bytes
9   0.00038 X.X.176.144 -> Y.Y.252.47 IP  D=Y.Y.252.47 S=X.X.176.244LEN=140, ID=0
9   0.00038 X.X.176.144 -> Y.Y.252.47 UDP D=1023 S=2049 LEN=120
9   0.00038 X.X.176.144 -> Y.Y.252.47 RPC R (#8) XID=890970798 Success
9   0.00038 X.X.176.144 -> Y.Y.252.47 NFS R GETATTR3 OK
```

The two highlighted numbers from the tcpdump output are also highlighted in the snoop output. But strangely enough, snoop identifies these as RPC transaction IDs. A quick search of Google did not reveal much with regards to this. I concluded that somehow tcpdump is misinterpreting the transaction ID as the destination port. As a sidenote, snoop and tcpdump cannot natively read one another's files. A tool like Ethereal (http://www.ethereal.com) can be used to open both and save them in different formats.

I have the luxury of knowing what the layout of network is around this sensor as it was implemented by some of my co-workers. Below is a simple diagram. The two hosts are on different routed segments. Routing between the two vlans is done on the MSFC (routing module) of the Cisco Catalyst 6509. A span (mirror) port is set up on the Catalyst which takes in/out traffic from both vlans and send them to the Sourcefire IDS.



Sourcefire

This sensor was implemented to monitor traffic within our network. Because of that, rules implemented (or Active) on the Sourcefire are few as our IDS team (person) is in the process of getting it tuned so that he is not drowning in data. As an example, attached below (next page) is the "Event Statistics" from our sensors (Graphic 1). It is painfully obvious that the number of alerts over the 10-day period is overwhelming. The interesting signature itself had seen 2500 hits in a ten-day period.

2. Detect was generated by:

The versions of the sensor were as follows: Management Console v2.7.0 (build 13) running on Sourcefire Linux OS v3.0.0.

The specific signature was:
Snort Rule #1322 ACTIVE
alert ip $EXTERNAL_NET any -> $HOME_NET any
msg:"BAD-TRAFFIC bad frag bits" fragbits:MD sid:1322 classtype:misc-activity rev:5

This rule is unidirectional (identified by the -> operator) and applies to IP traffic from the networks defined with the $EXTERNAL_NET variable to networks defined by the $HOME_NET variable. It specifically alerts on packets that have the DF bit and the MF bits set. The Snort SID states that the expected response for this type of packet would be an ICMP Type 3 Code 4 (it actually states type 5 Source Route Failed, but I believe that to be a mistake), Destination Unreachable - Fragmentation needed and don't fragment was set (more on this later).

3. Probability the source address was spoofed:

As in the first detect above, I used p0f to do some passive OS fingerprinting. Below is a TCP SYN packet from the tcpdump data collected above from the source and the destination:

```
00:36:30.020904 X.X.176.144.32789 > Y.Y.252.47.ssh: S [tcp sum ok]
3790844184:3790844184(0) win 5840 <mss 1460,sackOK,timestamp 11790190 0,nop,wscale 0>
(DF) (ttl 64, id 22134, len 60)
00:42:09.653709 Y.Y.252.47.45497 > X.X.176.144.ssh: S [tcp sum ok]
2067802396:2067802396(0) win 24820 <nop,nop,sackOK,mss 1460> (DF) (ttl 62, id 63114,
len 48)
```

We will use the Window Size, TTL, MSS and LEN fields to create a list of possible hosts from the p0f host table (http://www.stearns.org/p0f/p0f.fp)

Window Size	TTL	MSS	DF	WS	sackOK	nop	LEN	OS Description
SOURCE Possibilities								
5840	64	1460	1	0	1	1	60	Linux 2.4.2 - 2.4.14 (1)
5840	64	1460	1	0	0	1	60	Linux 2.4.13-ac7
5840	64	1460	0	0	1	1	60	Linux (unknown?) (2)
5840	64	1460	1	223	1	1	60	Linux-2.4.13-ac7
DESTINATION Possibilities								
24820	64	1460	1	0	0	1	48	SCO UnixWare 7.1.0 x86 ? (3)
24820	64	1460	1	-1	1	1	48	SunOS 5.8

We have narrowed it down to a some version of Linux for the source and either SCO Unix or SunOS 5.8 (Solaris 8) for the destination. So now we can use the DF flag and the sackOK flag to make our final determination. The source was Linux 2.4.2-2.4.14 (1) and the destination was SunOS 5.8.

SOURCEfire **Management Console** SOURCEfire

EVENTS | RULES | STATUS | ADMIN | LOGOUT Help

Sensor Stats Time: June 24, 2003, 7:38 am

Time Range 06/24/03 06:35 – 06/24/03 07:35 Change Time Reset

Sensor IP	Alerts/Sensor (in timespan)	Total Alerts/Sensor	Percentage of Total Alerts
TOTAL	383507	15966359	
[redacted]47.225	337446	6389829	(5%)
[redacted]47.226	21531	2979179	(1%)
[redacted]47.227	5463	770507	(1%)
[redacted]47.228	19067	5826844	(0%)

.../snip/...

Top Alerts

Alert Message	Count
spp_stream4: NULL Stealth Scan	158954
spp_stream4: Stealth Activity Detected	146044
SCAN UPnP service discover attempt	37532
spp_stream4: Retransmission	10526
SNMP request udp	7054
ICMP Destination Unreachable (Communication Administratively Prohibited)	4790
spp_stream4: VECNA Stealth Scan	2940
ICMP L3retriever Ping	2794
BAD-TRAFFIC bad frag bits	2500
RPC portmap proxy attempt UDP	1178
spp_stream4: NMAP XMAS Stealth Scan	1134
WEB-FRONTPAGE posting	1101
spp_stream4: SYN FIN Stealth Scan	982
spp_stream4: Full XMAS Stealth Scan	817
spp_stream4: SAPU Stealth Scan	655
spp_stream4: FIN Stealth Scan	650
spp_stream4: Window Violation	597
RPC mountd UDP unmount request	565
NETBIOS NT NULL session	491
RPC portmap NFS request UDP	396

Graphic

3. Probability the source address was spoofed (continued):

I did a search of our internal network managers listing for these IP addresses. Fortunately both were associated with live human beings!! As with many static lists and databases, the records can sometimes be incomplete and dated. I contacted them and asked permission to do a quick scan for the hosts and both agreed. Below is the output:

```
giac# nmap -O X.X.128.59

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not find at least 1
open and 1 closed TCP port
Interesting ports on XXXXXXXX (X.X.128.59):
(The 1600 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
Remote operating system guess: Linux Kernel 2.4.0 - 2.5.20
Uptime 1.386 days (since Mon Jun 23 15:39:37 2003)

------------

giac# nmap -O Y.Y.252.47

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on YYYYYYYY (Y.Y.252.47):
(The 1588 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
25/tcp     open        smtp
79/tcp     open        finger
111/tcp    open        sunrpc
512/tcp    open        exec
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
4045/tcp   open        lockd
32771/tcp  open        sometimes-rpc5
32772/tcp  open        sometimes-rpc7
Remote operating system guess: Solaris 8 early access beta through actual release
Uptime 112.762 days (since Tue Mar  4 06:39:59 2003)

Nmap run completed -- 1 IP address (1 host up) scanned in 54 seconds
```

As we can see, nmap's guess matched my guess based on some of the collected packets from these two systems. It is interesting that nmap seemingly accurately identified the source host with only one open port. I will mention again a very interesting post from Ofir Arkin regarding Linux Kernel 2.4.x and IP ID values (http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1)

My conclusion would be that these hosts were not spoofed. They are live hosts on our internal network. More importantly, the question now is what is causing this traffic and is one of these systems compromised?

4. Description of attack:

Below is the alerts from the Sourcefire sensor. It was expored as a csv file:

Message	Timestamp	Sensor IP	IP Src	IP Dst	Src Prt	Dst Prt	Class	Priority	Proto
BAD-TRAFFIC bad frag bits	5/24/2003 0:08	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:08	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:06	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:06	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:05	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:05	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:03	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:03	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP
BAD-TRAFFIC bad frag bits	5/24/2003 0:01	Z.Z.47.226	X.X.176.44	Y.Y.252.47	0	0	Misc activity	3	UDP

A detail of a one of these packets is attached below:

Packet Display

Resolve IP	Display Session

Event Data

Message	BAD-TRAFFIC bad frag bits	Generator ID	1
Classification	Misc activity	Snort ID	1322
Priority	3	Revision	5
Rule	alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC bad frag bits"; fragbits:MD; sid:1322; classtype:misc-activity; rev:5;)		Deactivate

Ethernet Header

SRC MAC:	00:D0:05:40:13:FC	DST MAC:	08:00:20:96:1E:62	Type:	0x0800

IP Header

Version:	4	Header Len:	5	TOS	0	Total Len (in bytes)	1500
16-bit ID	41931	Frag Flags	DF MF	13-bit offset		0x00B9 (185)	
TTL	62			Protocol		UDP	
Source IP				X.X.176.144			
Destination IP				Y.Y.252.47			

I would not classify this as an attack yet. From the tcpdump file we notice that this are fragmented UDP NFS packets. The thing that sets them apart from being completely normal is that the DF and the MF bits are both set. It is possible that there is some sort of vulnerability that

the source station is looking to exploit. All this assumes that either the user on the source station has malicious intent or the system has been compromised. For argument's sake, I will assume the user (an employee at our facility), would not partake in such terrible activities ☺.
From the information we have already gathered, we know that only TCP port 22 (SSH) is listening on source system. It is possible that an SSH vulnerability was used to get into the source. I wouldn't call this an OS fingerprinting attempt because from our nmap scan of the destination host, that information was too easily available to go through this type of trouble. If this was an attack, it was a specifically targeting something on the system.

X.X.176 red hat
Y.Y.252.47 solaris 2.8

5. Attack mechanism & 6. Correlations:
A search of Google was in order at this point. I was specifically looking for information about UDP NFS packets with both the DF and MF bits set. Interestingly enough I ran across this following writeup:
http://kerneltrap.org/node.php?id=579
and again on RedHat's site:
https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=58084

A post on the RedHat Bugzilla page for this problem quoted the RFC (http://www.faqs.org/rfcs/rfc1191.html):

> Also, since a single NFS operation cannot be split across several UDP datagrams, certain operations (primarily, those operating on file names and directories) require a minimum datagram size that may be larger than the PMTU. NFS implementations should not reduce the datagram size below this threshold, even if PMTU Discovery suggests a lower value.(Of course, in this case datagrams should not be sent with DF set.)

These are apparently a well-known set of packets that are easily reproducible. It was also apparent was a split as to what the right way of doing things was.

7. Evidence of active targeting:
This was determined to be legitimate NFS traffic from a Linux system to a Solaris system. It is the NFS implementation on Linux that seems to be generating these packets. After the above analysis was completed, the two network managers were asked to reproduce the connection for the purposes of data capture to verify that this was legitimate traffic. It was determined to legitimate traffic.

8. Severity:
The formula:
severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality – 1, this was legitimate traffic.
Lethality - 1, this was legitimate traffic.

System countermeasures - 3, although this was legitimate traffic the destination host had many listening TCP ports. We do not know if it had TCPwrappers or the equivalent for securing the system. The source host, on the other hand, seemed have very few listening ports (only 1, but that was a limited portscan).
Network countermeasures – 2, from the portscan of the destination host, it was apparent that quite a bit was being permitted into that network.

Severity = (1 + 1) - (3 + 2)
Severity = -3

9. Defensive recommendation:
This was not an attack, but, I would still recommend that the destination host implement some sort of host-based firewall. It is possible that TCPwrappers (or something like it) has been implemented on the system, but we don't know that information. I would also recommend that depending on the needs, some things be filtered from coming into these network. Firewalls or firewall type functionality should be implemented to better protect the systems within.

10. Multiple choice test question:
All fragmented packets, except for the last fragment, should have datagram length of _____.

A) 1500
B) 1480
C) MTU size of the medium on which it is traveling.
D) MTU of the source host
E) It is variable.
Answer: A) 1500

DETECT 3

1. Source of Trace:

This trace was from my work network.
The alerts were collected from Cisco IDS sensors (www.cisco.com). The Cisco IDS has the capability to capture certain signatures into binary capture files. Opening these files can sometimes be tricky. eEye's Iris works well at doing this (www.eeye.com). Ethereal, tcpdump, and windump were unable to open the binary file. The alert that caught my attention was a CDE buffer overflow attack known as "CDE dtspcd overflow". CDE, or Common Desktop Environment, is a window manager for X Windows (http://www.opengroup.org/cde). This alert triggers when the sensor detects a buffer overflow attempt to the CDE daemon (dtspcd) on TCP port 6112. The priority level of this signature is a 5, which on Cisco sensor is high. The following is the explanation of the signature for Cisco:

Exploit Signature

CDE dtspcd overflow			
ID: 3700		**Sub ID:** 0	
Default Alarm Level:	HIGH (5)	**Signature Type:**	NETWORK
Signature Structure:	COMPOSITE	**Implementation:**	CONTENT
Release Version:	S15		

Description: This signature will fire if a buffer overflow attack to the CDE sub-process control daemon (dtspcd) on TCP port 6112 is detected.

Benign Trigger(s): No known triggers.

Recommended Signature Filter: No recommended filters.

Data Field Information Tag: None

Related Vulnerabilities: 1003700

User Notes: User Notes Page

I did find a two CERT advisories on this particular type of attack.
http://www.cert.org/advisories/CA-2002-01.html
http://www.cert.org/advisories/CA-2001-31.html

The newer advisory, CA-2002-01, has a specific hex packet dump and information on what to look for exactly. From CA-2002-1:

```
The signature can be found at bytes 0x3e-0x41 in the following attack packet from a
tcpdump log (lines may wrap):

09:46:04.378306 10.10.10.1.3592 > 10.10.10.2.6112: P 1:1449(1448) ack 1 win
16060 <nop,nop,timestamp 463986683 4158792> (DF)
0x0000   4500 05dc a1ac 4000 3006 241c 0a0a 0a01        E.....@.0.$.....
0x0010   0a0a 0a02 0e08 17e0 fee2 c115 5f66 192f        ...f........_f./
0x0020   8018 3ebc e1e9 0000 0101 080a 1ba7 dffb        ..>.............
0x0030   003f 7548 3030 3030 3030 3032 3034 3130        .?uH0000000204 10
0x0040   3365 3030 3031 2020 3420 0000 0031 3000        3e0001..4....10.
0x0050   801c 4011 801c 4011 1080 0101 801c 4011        ..@...@.......@.
0x0060   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.

The value 0x103e in the ASCII (right) column above is interpreted by the server as the
number of bytes in the packet to copy into the internal 4K (0x1000) buffer. Since
0x103e is greater than 0x1000, the last 0x3e bytes of the packet will overwrite memory
after the end of the 4K buffer.
```

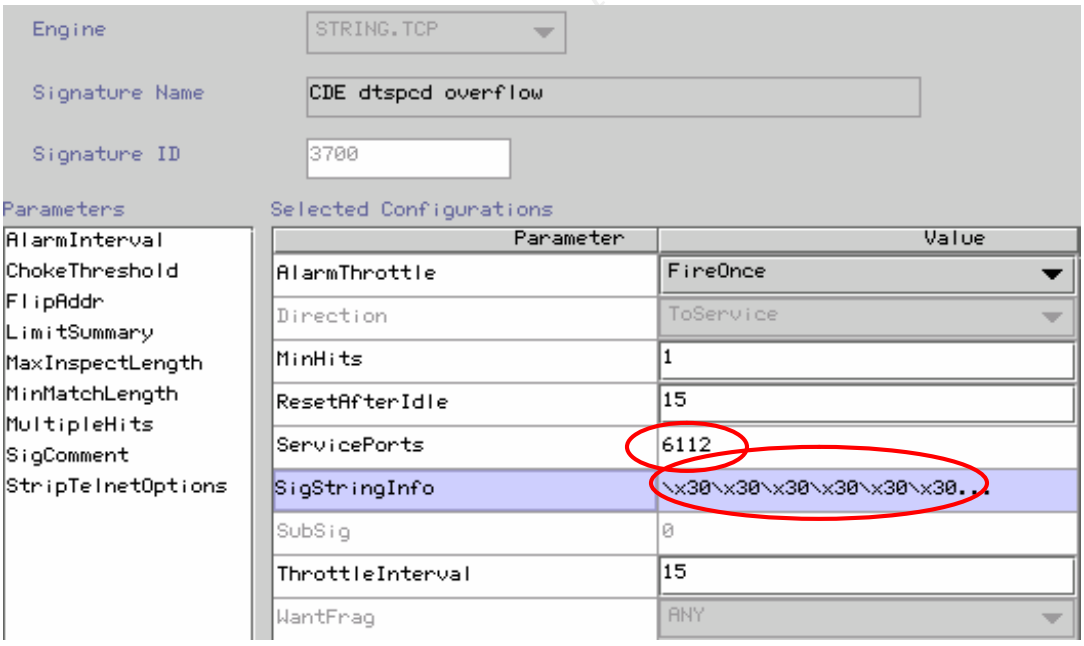
Iris was used to read the file created from the Cisco IDS and then dump the entire packet to a text file. Below is the packet dump of the packet that triggered this signature in hex along with a packet back from the destination:

```
Timestamp:            12:14:02:046
MAC source address:   00:D0:00:2C:63:FC
MAC dest address:     00:0B:46:B3:6E:00
Frame type:           IP
Protocol:             TCP->DTSPCD
Source IP address:    X.X.105.131
Dest IP address:      210.242.84.67
Source port:          57424
Destination port:     6112
SEQ:                  711286895
ACK:                  2576160
Packet size:          1514

Packet data:
0000: 00 0B 46 B3 6E 00 00 D0 00 2C 63 FC 08 00 45 00 ..F.n....,c...E.
0010: 05 DC A3 D8 40 00 3E 06 81 C6 XX XX XX XX D2 F2 ....@.......i...
0020: 54 43 E0 50 17 E0 2A 65 60 6F 00 27 4F 20 50 18 TC.P..*e`o.'O P.
0030: C1 E8 02 EC 00 00 F7 43 B4 05 09 03 01 00 00 00 .......C........
0040: 00 00 00 00 5A 3A 2D EA 48 4D 33 57 00 00 00 00 ....Z:-.HM3W....
0050: 7C 63 30 30 30 30 30 30 66 66 5A 6F 61 74 6F 72 |c000000ffZoator
0060: 20 54 44 20 32 2E 30 7C 72 00 70 0D 00 00 0A 00  TD 2.0|r.p.....
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
========================// SNIP//====================================
0570: 97 28 49 26 91 A2 44 6A 25 4A F0 4E C1 A2 09 3A .(I&..Dj%J.N...:
0580: 92 A5 AF 25 9C 28 B4 03 3D 13 D0 93 E2 70 0A 70 ...%.(..=....p.p
0590: A8 BA C4 A1 0F 16 A1 DB 39 42 97 18 5C 0D 42 0A ........9B..\.B.
05A0: B8 D7 5D 10 9F 57 1E DA E5 E1 AC 7F 15 BC 4C AD ..]..W........L.
05B0: 4F B9 FD 1F 8A 48 0E C7 C7 DD DE 4B 28 6B 0E 0A O....H.....K(k..
```

```
05C0: 2E 50 60 06 B2 0E 40 4F 8D 28 15 DE CD A1 E4 31 .P`...@O.(.....1
05D0: 28 00 BB E1 14 32 1D 58 E2 F3 03 F7 7A 29 20 94 (....2.X....z) .
05E0: CA 50 88 16 60 11 56 0A 5D 22                    .P..`.V.]"

=====================================================================

Timestamp:            12:14:02:046
MAC source address:   00:0B:46:B3:6E:00
MAC dest address:     00:00:0C:07:AC:01
Frame type:           IP
Protocol:             TCP->DTSPCD
Source IP address:    210.242.84.67
Dest IP address:      X.X.105.131
Source port:          6112
Destination port:     57424
SEQ:                  2576160
ACK:                  711288355
Packet size:          64

Packet data:
0000: 00 00 0C 07 AC 01 00 0B 46 B3 6E 00 08 00 45 00 ........F.n...E.
0010: 00 32 0F 60 40 00 6D 06 EC E8 D2 F2 54 43 XX XX .2.`@.m.....TC..
0020: XX XX 17 E0 E0 50 00 27 4F 20 2A 65 66 23 50 18 i....P.'O *ef#P.
0030: 22 38 9E BD 00 00 F7 45 0A 00 03 09 01 00 00 00 "8.....E........
0040:
```

2. Detect was generated by:

The sensor was a Cisco 4230 running version 3.1(3)S46 code. The following is the actual signature that the Cisco IDS used to trigger this event:



The highlighted field called "SigStringInfo" is what the IDS is looking for to trigger this alert along with a TCP port of 6112. The rule is basically looking for any TCP port 6112 traffic with the above pattern in it (circled in red). This detected was generated by a host on my network X.X.105.131 and was destined for Y.Y.84.67 on a remote network.

3. Probability the source address was spoofed:

To do this, I first want to gather some information about the destination. I already know the source is on my local network.

```
Liu, Hui-Min
   No. 27-3, Lin Shen Rd,
   Pingtung Taiwan
   TW

   Netname: LIU-HUI-MIN-PT-NET
   Netblock: 210.242.84.0 - 210.242.84.255

   Administrator contact:
      Hui Min Liu (HML2-TW) hn80090325@hn.hinet.net
      TEL: +886-8-722-8771

   Technical contact:
      Hui Min Liu (HML2-TW) hn80090325@hn.hinet.net
      TEL: +886-8-722-8771
```

Interestingly enough, usually attacks are being sourced from Asian networks. In this case, they seem to be the target.

This particular attack is very targeted. The attacker would not gain much if his source address was spoofed and he was attempting a buffer overflow. That said, I tried some basic connectivity testing to see if this host was alive:

```
gcia# ping X.X.105.131
PING X.X.105.131 (X.X.105.131): 56 octets data
64 octets from X.X.105.131: icmp_seq=0 ttl=255 time=1.5 ms
64 octets from X.X.105.131: icmp_seq=1 ttl=255 time=0.4 ms
64 octets from X.X.105.131: icmp_seq=2 ttl=255 time=0.4 ms
64 octets from X.X.105.131: icmp_seq=3 ttl=255 time=0.4 ms

--- 128.196.128.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.6/1.5 ms
```

So this host is alive on the network. Based on this data and the type of exploit that we are looking at. I would be fairly comfortable saying the source is not spoofed.

4. Description of attack:

I followed the example set in the CERT advisory to manually look for this particular exploit in the packet dump. Below is part of the initial packet that is believed to have the malicious data in it.

```
Timestamp:            12:14:02:046
MAC source address:   00:D0:00:2C:63:FC
MAC dest address:     00:0B:46:B3:6E:00
Frame type:           IP
Protocol:             TCP->DTSPCD
Source IP address:    X.X.105.131
Dest IP address:      210.242.84.67
Source port:          57424
Destination port:     6112
```

```
SEQ:                    711286895
ACK:                    2576160
Packet size:            1514

Packet data:
0000: 00 0B 46 B3 6E 00 00 D0 00 2C 63 FC 08 00 45 00 ..F.n....,c...E.
0010: 05 DC A3 D8 40 00 3E 06 81 C6 XX XX XX XX D2 F2 ....@.......i...
0020: 54 43 E0 50 17 E0 2A 65 60 6F 00 27 4F 20 50 18 TC.P..*e`o.'O P.
0030: C1 E8 02 EC 00 00 F7 43 B4 05 09 03 01 00 00 00 .......C........
0040: 00 00 00 00 5A 3A 2D EA 48 4D 33 57 00 00 00 00 ....Z:-.HM3W....
0050: 7C 63 30 30 30 30 30 30 66 66 5A 6F 61 74 6F 72 |c000000ffZoator
0060: 20 54 44 20 32 2E 30 7C 72 00 70 0D 00 00 0A 00  TD 2.0|r.p.....
```

Looking at the alert, it's pretty apparent why this packet triggered that signature. I've highlighted the HEX responsible for triggering the alert. A similar set of 3030 3030 3030 appears in the CERT advisory example. According to the CERT advisory, the actual buffer overflow portion of the attack packet is found at bytes 0x3e – 0x41. I've highlighted those bytes in our packet above. It seems as though this packet does not match the attack packet described in the advisory. The buffer size is 4K (0x1000). We were looking for a value of something larger than that (0x103e was the CERT sample). If the packet was larger than 0x1000, the difference between the values would overwrite memory after the end of the 4K buffer. I would have to come to the conclusion that this is a false positive. I would say that the signature that the Cisco IDS is using is too broad and needs to be tuned.

5. Attack mechanism

The attack mechanism, in this case, would have been the 4K buffer being overwritten by data larger than 4K that would overwrite memory after that buffer. According to SecurityFocus.com (http://www.securityfocus.com/bid/3517/info) with regards to this vulnerability, the overflow is in the libDtSvc library. The 'Subprocess Control Service' uses this library. The vulnerability is then exploited via the 'dtspcd' service on the end station.

Unfortunately (well, fortunately for the destination host, but unfortunately for the analysis of this detect, this was a false positive. The signature is a bit too general and should be more specific as to which bytes it is searching for. The byes are well known and documented. It should be a simple of matter of adjusting the rule to cut down this false positive.

6. Correlations:

Even though this was determined to be a false positive alert, I still had some interest in seeing whether this internal host was the culprit of other malicious activities. Whether this was a buffer overflow or not, I question the need to connect to the dtspcd service on a system in Taiwan. I did a search of the usual suspects, Dshield, myNetWatchman, and DeepSight Analyzer (Security Focus). Nothing turned up on Dshield but I did find some very interesting on the other two sites. MyNetWatchman had a record of this host attempting the same buffer overflow to another external host. Below is the record:

Most Recent Event Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of IPs Targeted	IP Protocol	Target Port	Port/ Issue Description	Source Port	Explanation	Event Count

25 Jun 2003 04:02:54	Readytek	Perl	Cisco PIX	63.226.x.x	1	6	6112	dtspcd CDE Buffer Overflow	56302	mNW Info	5

A search of DeepSight Analyzer gave a report of several types of malicious activity from this host.

IP Analysis

Jun 18 2003 - Jun 25 2003

SecurityFocus DeepSight™ Analyzer

IP Address	[redacted]105.131	**Host Name**	[redacted]
Country	United States		
Last Event Date	Jun 25 2003		
DeepSight Users Affected	5		
Users that have notified	0		

Domain Contact

[redacted]

ISP Contact

Various Registries (Maintained by ARIN)

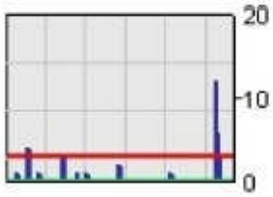
United States

IDS Event Summary

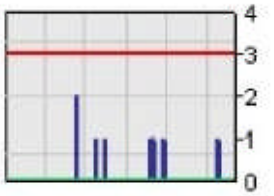
Microsoft IIS/PWS Escaped Characters Decoding Command

Microsoft IIS 5.0 .printer ISAPI Extension Buffer Overflow Attack

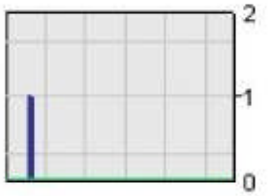
Apache Server Side Include Cross Site Scripting Attack



Generic CDE dtspcd Buffer Overflow Attack



Generic Cross-Site Scripting in URL Attack



Type of Event	24 Hrs	7 Days	Sensors
Unknown Attack	0	184	Default
Generic TCP Half Open (Syn Scanning) Portscan Probe	8	135	Default
Microsoft IIS 5.0 .printer ISAPI Extension Buffer Overflow Attack	22	50	Default
Microsoft IIS/PWS Escaped Characters Decoding Command Execution Attack	130	213	Default
Generic CDE dtspcd Buffer Overflow Attack	1	8	Default
Apache Server Side Include Cross Site Scripting Attack	21	31	Default

IDS Port Summary

Ports		Events
80	TCP / http / www World Wide Web HTTP	363
6112	TCP / dtspcd CDE subprocess control	8

The fact that this is a false positive seems to be the exception and the rule in this case. It's pretty apparent that this host has been involved in some malicious activity in the very recent past. Luckily, my hunch about CDE traffic destined for Taiwan was right, otherwise I would have wasted my time analyzing something that did not need any more analysis.

8. Severity:

The formula:
severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality – 2 – This did not turn out to be the buffer overflow attempt, but it may very well have been a reconnaissance attempt to see if the remote host would answer on that port.

Lethality – 3 – If we only look at the fact that this was a false positive, I would have said a 1. The results from DeepSight Analyser and myNetWatchman, added to the thought that this may have been a recon attempt, (a dry run of sorts) put this category up to a three for me.

System Countermeasures – 1 – In most normal cases, a host in Taiwan should not be responding to CDE traffic from the United States, and vise versa.

Network Countermeasures – 1 – This traffic was never filtered out, from source to destination. I would think that most networks would not want this type of traffic coming into their enterprise. Systems across the globe do not need to communicate with CDE on my host.

Severity = (2 + 3) - (1 + 1)
Severity = 3

9. Defensive recommendation:

First, the remote system should be set up with some sort of protection that would block or refuse a TCP port 6112 connection attempt from across the world. The host would appear to be running with little or no control if this service is left listening. Next, I would expect the network of the remote system to not permit TCP 6112 CDE traffic in to its internal network. This should be one of the things that gets blocked at the border along with all the other "never leave the borders" type rules.

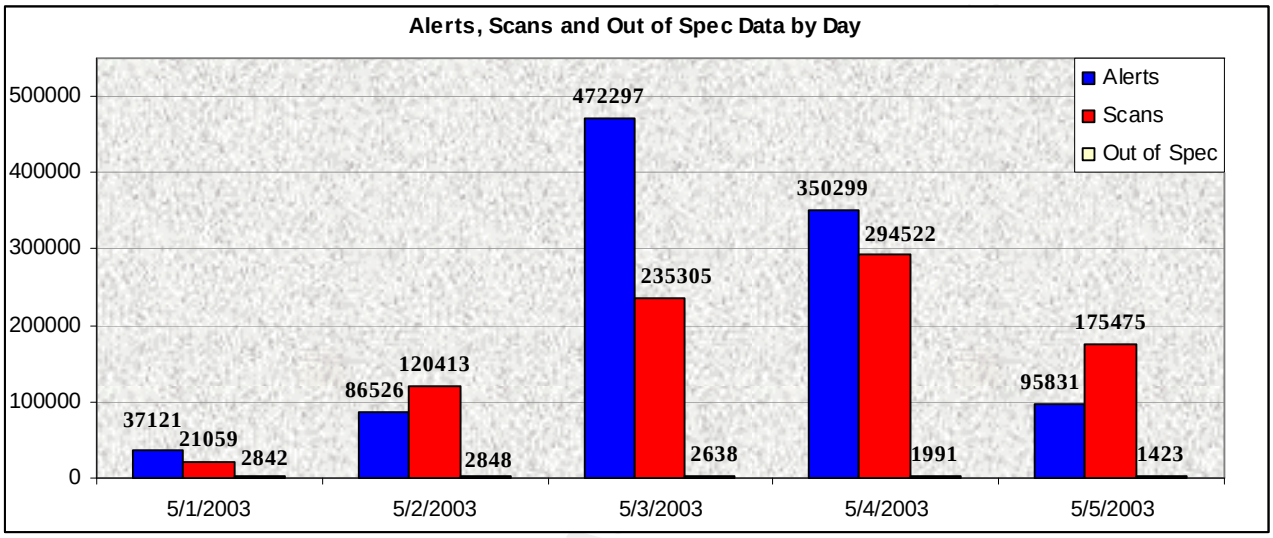
10. Multiple choice test question:

This particular signature caused a false positive. What could have been added to the signature to help reduce the chance of it re-occurring?
A. Nothing, false positives are fact of life.
B. The rule should have searched for the 0x1000 buffer in the data
C. The rule should have searched for a value larger than 0x1000 in the buffer
D. The rule should have used the known start byte field of the signature
Answer: D

ANALYZE THIS

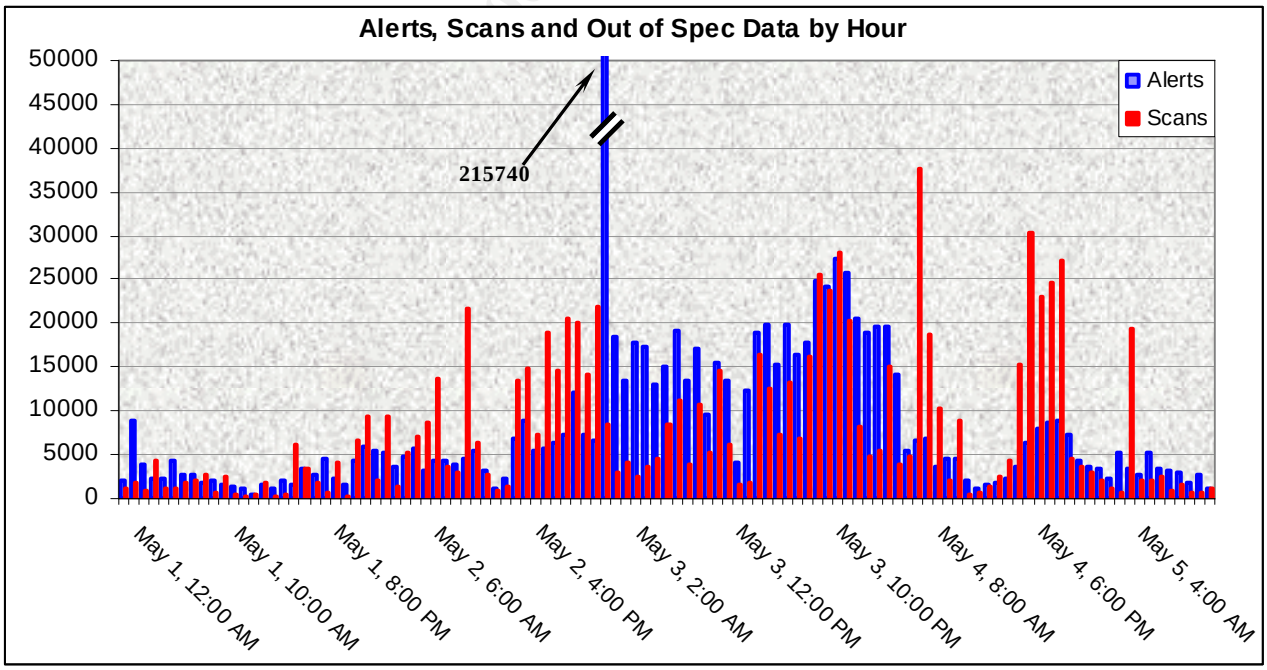
EXECUTIVE SUMMARY - THE HEAVY HITTERS

Over the five day period, there were nearly 1.9 (1,888,848) million events. There was 1,042,074 alerts and 846,774 scanning entries. Of these events, 11,742 Out of Spec packets were detected. The following graph (Graph 1) better illustrates this over the five days.



Graph 1

To see the trends with better detail, I also looked at the number of events per hour (Graph 2).



Graph 2

At first glance, this is a tremendous amount of alerts. It seems that the sensor rules need to be tuned so the sheer noise level is reduced. With this much noise, it is difficult to do proper analysis of an event. In fact, it events could very easily be missed. I will go through the top 93% of these alerts to help tune this sensor and get it to a state where the alerts we see are less often false positives. I will start with the heavy hitters in the alerts files and will use the scan and OOS file to correlate data and help in the analysis of this sensor.

LOG FILES

There was three different types of log files that were provided for analysis. I chose to use the files dated from May 1st through May 5th. The following table (Table 1) provides some basic details about the files:

Alert Files	Scan Files	Out of Spec Files
alert.030501.gz	scans.030501.gz	OOS_Report_2003_05_02_28431.txt
alert.030502.gz	scans.030502.gz	OOS_Report_2003_05_03_7239.txt
alert.030503.gz	scans.030503.gz	OOS_Report_2003_05_04_21395.txt
alert.030504.gz	scans.030504.gz	OOS_Report_2003_05_05_25821.txt
alert.030505.gz	scans.030505.gz	OOS_Report_2003_05_06_7938.txt

Table 1

The log files were generated by Snort with a standard rulebase. I chose to analyze all the logs as complete sets. The five files of each type from above were merged into one each. For simplicity, I called them all.alerts, all.scans, and all.oos. There was a strange anomaly that I noticed in the date stamp of the data from the OOS (Out of Spec). It seems the records with a file are one day behind the date of the file itself. For example, to get OOS data for May 5th, I used the OOS file dated May 6th.

When creating the all.alerts file, all the port scan entries (clearly labeled with spp_portscan) were filtered out. The following is a sample of a record in the alerts file:

```
05/01-11:20:22.757565  [**] SMB Name Wildcard [**] 165.247.107.139:137 ->
MY.NET.24.34:137
05/01-11:31:01.381487  [**] SMB Name Wildcard [**] 148.220.29.18:1033 ->
MY.NET.56.28:137
05/01-11:20:22.793455  [**] SMB Name Wildcard [**] 67.31.249.251:137 ->
MY.NET.24.34:137
```

There were also a large number of malformed entries in the alerts logs that were scraped out with Unix text manipulation tools like grep and tr so that all the data was normalized.

The format of the scan file records was as follows:

```
May  1 11:20:47 69.22.247.251:2648 -> MY.NET.130.228:445 SYN ******S*
May  1 11:31:40 MY.NET.1.3:32832 -> 212.100.224.80:53 UDP
```

```
May  1 11:31:40 MY.NET.1.3:32832 -> 209.208.92.254:53 UDP
```

The format of the OOS files was as follows:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/01-00:06:20.935520 133.11.36.55:47976 -> MY.NET.24.34:80
TCP TTL:53 TOS:0x0 ID:17430 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x42F1E07D  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 111256683 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/01-00:06:21.575096 133.11.36.35:47532 -> MY.NET.24.34:80
TCP TTL:53 TOS:0x0 ID:7221 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x45280763  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 111287126 0 NOP WS: 0
```

A custom Perl script was written to help with the formatting of this file to so it could be processed and pertinent information gained from it [OOS Scripts].

Alerts

The table (Table 2) below is a numerically sorted listing of all the alerts from the alerts file.

Percent of Total	Count	Alert
39.4033%	355527	Incomplete Packet Fragments Discarded
23.1139%	208552	TCP SRC and DST outside network
19.3096%	174226	SMB Name Wildcard
3.3738%	30441	spp_http_decode: IIS Unicode attack detected
3.0232%	27278	High port 65535 udp - possible Red Worm - traffic
2.7646%	24944	CS WEBSERVER - external web traffic
2.6201%	23641	High port 65535 tcp - possible Red Worm - traffic
1.5005%	13539	Tiny Fragments - Possible Hostile Activity
1.0352%	9340	TFTP - Internal TCP connection to external tftp server
0.6676%	6024	EXPLOIT x86 NOOP
0.5580%	5035	connect to 515 from outside
0.5569%	5025	[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
0.5564%	5020	spp_http_decode: CGI Null Byte attack detected
0.2743%	2475	Null scan!
0.1748%	1577	Queso fingerprint
0.1732%	1563	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
0.1490%	1344	MY.NET.30.4 activity
0.1021%	921	Possible trojan server activity
0.0891%	804	MY.NET.30.3 activity
0.0866%	781	CS WEBSERVER - external ftp traffic
0.0827%	746	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
0.0795%	717	IDS552/web-iis_IIS ISAPI Overflow ida nosize
0.0576%	520	SUNRPC highport access!
0.0438%	395	TFTP - Internal UDP connection to external tftp server

0.0300%	271	[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot
0.0215%	194	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
0.0208%	188	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
0.0186%	168	IRC evil - running XDCC
0.0165%	149	External RPC call
0.0165%	149	[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
0.0161%	145	NMAP TCP ping!
0.0142%	128	EXPLOIT x86 setuid 0
0.0109%	98	SNMP public access
0.0066%	60	NIMDA - Attempt to execute cmd from campus host
0.0059%	53	EXPLOIT x86 setgid 0
0.0057%	51	EXPLOIT x86 stealth noop
0.0038%	34	TCP SMTP Source Port traffic
0.0029%	26	Back Orifice
0.0029%	26	Notify Brian B. 3.54 tcp
0.0024%	22	Notify Brian B. 3.56 tcp
0.0014%	13	SMB C access
0.0013%	12	Probable NMAP fingerprint attempt
0.0011%	10	Attempted Sun RPC high port access
0.0009%	8	RFB - Possible WinVNC - 010708-1
0.0008%	7	FTP passwd attempt
0.0008%	7	[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.
0.0007%	6	TFTP - External UDP connection to internal tftp server
0.0004%	4	DDOS shaft client to handler
0.0003%	3	NIMDA - Attempt to execute root from campus host
0.0003%	3	TFTP - External TCP connection to internal tftp server
0.0002%	2	EXPLOIT x86 NOPS
0.0002%	2	SYN-FIN scan!
0.0001%	1	Bugbear@MM virus in SMTP
0.0001%	1	DDOS TFN Probe
0.0001%	1	[UMBC NIDS IRC Alert] Possible trojaned machine detected
0.0001%	1	site exec - Possible wu-ftpd exploit - GIAC000623
	902,278	TOTAL ALERTS

Table 2

We can see from this data that the first three items make up a majority of the alerts.

355527	Incomplete Packet Fragments Discarded
208552	TCP SRC and DST outside network
174226	SMB Name Wildcard
30441	spp_http_decode: IIS Unicode attack detected
27278	High port 65535 udp - possible Red Worm - traffic
24944	CS WEBSERVER - external web traffic
23641	High port 65535 tcp - possible Red Worm - traffic

In fact, they make up nearly 82% of the alerts (738,305 of the 902,278 total). We will take a quick look at each of the above alerts.

Incomplete Packet Fragments Discarded
Background:

These alerts are not a result of a rule. They come from Snort's de-fragmentation preprocessor. According to Dragos Ruiu, the person who wrote the preprocessor, the alert is sent when "…packets bigger than 8k that are more than half empty when the last fragment is received are discarded." (Ruiu, http://www.geocrawler.com/archives/3/4890/2001/2/350/5151528). Some of the caused Dragos gives for this sort of traffic is transmission errors, broken stacks, or a fragmentation attack. Also, according to Marty Roesch, older versions of the defrag preprocessor had a tendency to misbehave (http://archives.neohapsis.com/archives/snort/2001-11/0822.html).

Analysis:
In our alerts files, we found a total of 355,129 of these events from the source host MY.NET.210.114. Of these, 354,920 were sourced from port 0 and destined to port 0. Here is a sample of these alerts:

```
05/03-11:46:05.978952  [**] Incomplete Packet Fragments Discarded [**]
MY.NET.210.114:0 -> 213.97.198.23:0
05/03-11:46:06.535087  [**] Incomplete Packet Fragments Discarded [**]
MY.NET.210.114:0 -> 213.97.198.23:0
05/04-16:44:58.407868  [**] Incomplete Packet Fragments Discarded [**]
MY.NET.210.114:0 -> 213.97.198.23:0
05/04-16:44:58.765082  [**] Incomplete Packet Fragments Discarded [**]
MY.NET.210.114:0 -> 213.97.198.23:0
```

All the logs for these fragments were dated 5/3 and 5/4. There was no occurrence of this traffic pattern from this host on other days. For purposes of correlation, I searched the alert logs for anything from this host that was not one of the events seen above on those two days. Below are the results of that query:

```
05/03-13:17:00.074019  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.210.114:4589 -> 213.97.198.23:69
05/03-13:17:00.151041  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.210.114:4589 -> 213.97.198.23:69
05/03-13:17:00.160476  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.210.114:4589 -> 213.97.198.23:69


05/04-07:28:01.066431  [**] TFTP - Internal UDP connection to external tftp server
[**] MY.NET.210.114:1631 -> 213.97.198.23:69
05/04-13:07:33.787560  [**] TFTP - External UDP connection to internal tftp server
[**] MY.NET.210.114:69 -> 213.97.198.23:30698
```

These alerts identify the same two hosts from above participating in TFTP sessions on those two days.

In addition to this data, I searched the scan file to find some correlations. The internal host, MY.NET.210.114, and the external host, 213.97.198.23, seem to have been carrying on quite a conversation. The scan file shows 64,602 entries for UDP packets between these two hosts. Of these, 11,241 were sourced and destined for UDP port 0. The rest of the entries, 53,353, looked to be an old fashioned UDP port scan. A sample of each type of scan record follows:

```
May  3 11:54:00 MY.NET.210.114:4928 -> 213.97.198.23:42450 UDP
May  3 11:54:00 MY.NET.210.114:1610 -> 213.97.198.23:35487 UDP
```

```
May  3 11:54:00 MY.NET.210.114:0 -> 213.97.198.23:0 UDP
May  3 12:12:48 MY.NET.210.114:0 -> 213.97.198.23:0 UDP
```

Neither one of these hosts had any records in the OOS files.

Conclusion and Recommendations:
Although this particular signature has a tendency of being "noisy", from our logs we seem to have discovered some strange activity. This host is using TFTP (UDP port 69) to connect to an external host. In most circumstances, blocking all TFTP at the borders is strongly recommended. This doesn't seem to be the case on this network. This is a noisy alert that may be a false positive, but the existence of TFTP connections between these same two hosts during the same timeframe is a point of concern. I do not believe it is a false positive, I believe the source system should be investigated. As a side note, the TFTP rule seems to be a local rule as there is no entry for it in the current 2.0 Snort rules files.

This noise level from this host fortunately caused us to look at it in more detail. In many cases, rules that create this much noise usually get ignored. This source host should definitely be visited. It has either been compromised and is being used for scanning, or, the person using it is up to no good. Either way, it is something that needs to be looked at. I will gather some additional information about the external that seems to have been involved in some interesting traffic patterns on the internal network.

TCP SRC and DST outside network
Background:
This rule kicks off an alert when the sensor sees traffic that is sourced and destined for an outside network. In other words, an internal address is neither the source nor the destination of these packets.

Analysis:
Below are a few samples of the 208,552 alerts that this signature initiated:

```
05/02-22:17:31.471343  [**] TCP SRC and DST outside network [**] 0.0.0.0:49823 ->
140.99.99.99:80
05/03-02:01:10.435715  [**] TCP SRC and DST outside network [**] 192.168.2.100:4197 ->
216.149.164.100:80
05/03-11:47:35.456153  [**] TCP SRC and DST outside network [**] 18.173.203.23:1291 ->
216.200.173.18:6667
```

I searched through the scan and OOS files for any packets that didn't have a source or a destination of MY.NET.x.x and found none. From the sample of packets, it's obvious that some of the addresses are abnormal (0.0.0.0), some are private (192.168.2.100), and others are legitimate (18.173.203.23). Because of this variety of addresses, I decided to take a closer look at all the sources. A quick sort of all the above entries led me to an interesting observation. All the IP addresses (both source and destination) in these packets looked like [1-223].X.X.X, where the X seemed to be somewhat random but every network between 1 and 223 was used. It was not done sequentially, but after sorting the data by IP address the addressing/spoofing scheme was very obvious.

The next thing I looked for was the destination port of these packets. Of the 208,552 records, 205,501 were destined for TCP port 6667. This port is generally used for IRC (which is a sign of trouble as install IRC related services is very prevalent on compromised systems), but a quick search of the Neohapsis' ports list (http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html) revealed that there are many Trojans that utilize this port as well:

```
6667   tcp    [trojan] Dark FTP
6667   tcp    [trojan] EGO
6667   tcp    Internet Relay Chat
6667   tcp    IRCU
6667   tcp    Kaitex Trojan
6667   tcp    [trojan] Maniac rootkit
6667   tcp    [trojan] Moses
6667   tcp    [trojan] ScheduleAgent
6667   tcp    [trojan] ScheduleAgent
6667   tcp    [trojan] Subseven 2.1.4 DefCon 8
6667   tcp    [trojan] SubSeven
6667   tcp    [trojan] The Thing (modified)
6667   tcp    [trojan] Trinity
6667   tcp    [trojan] WinSatan
```

The scan file had a handful of entries for traffic destined to TCP port 6667, but it was not nearly at the level that we saw in the alerts. There was a total of 84 entries, a sample is below:

```
May  5 08:42:17 130.85.97.46:4304 -> 194.68.45.50:6667 SYN ******S*
May  5 09:06:14 130.85.97.46:2325 -> 194.68.45.50:6667 SYN ******S*
```

Also, there was one record in the OOS file that was directed at TCP port 6667. It was logged as Out of Spec because it had both ECN bits. We will talk about ECN a bit later in this document. Below is the record from the OOS file:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/05-02:27:13.576207 65.33.99.232:52126 -> MY.NET.70.198:6667
TCP TTL:45 TOS:0x0 ID:15406 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x5257372E  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 560794368 0 NOP WS: 0
```

Conclusion and Recommendations:
This is definitely spoofed traffic. We do not have enough information with regards to who/where the actual source host is, but it is likely a system on the internal network. I will make that statement with the caveat that assumes this network has proper ingress and egress filtering applied at the borders. From looking at the alert and scan files, I do not feel comfortable about statement. This traffic pattern may also be a compromised system or the user is using it for malicious purposes. I would definitely recommend that this problem be tracked down. Spoofed addresses are sometimes difficult to find, but not impossible. Depending on network layout, a tcpdump trace may be helpful in determining the source. It's unclear what is to be gained by this type of activity since the return traffic would never get back to the host that initiated it. The fact that it is targeted at a specific port that has a history of trouble is a cause for concern, hence the "malicious intent" label.

SMB Name Wildcard

Background:
This traffic pattern is usually normal windows traffic. It is usually a result of normal queries to determine NetBIOS names on a network. It would normally not be a concern, but these are all packets sourced from external networks. There has been some activity with regards to this port and traffic pattern being related to the "network.vbs" malicious VBScript. A good explanation of this traffic pattern and the some of the possible reasons behind it can be found at www.sans.org/resources/idfaq/port_137.php, in an Sans FAQ written by Bryce Alexander. Also, there is a CERT Incident Note (http://www.cert.org/incident_notes/IN-2000-02.html) on this particular topic. Although the patterns we see don't exactly match that CERT note, there is some distinct similarities. Even more information is available at Whitehats.com (http://whitehats.com/info/IDS177).

Analysis:
These alerts look very much like a targeted scan of the MY.NET.x.x/16 network. The third octet of all the destination addresses (MY.NET) increases sequentially. Also, the source addresses, when sorted, look like [1-221].[1-255].x.x. Once sorted, it is obvious that this is not normal traffic and is being spoofed. It seems that both the source and destination generated in some fashion and the packet is sent on its way. Below is a few sample packets:

```
05/01-11:20:21.688063  [**] SMB Name Wildcard [**] 218.14.155.3:11306 ->
MY.NET.2.212:137
05/01-11:20:22.757565  [**] SMB Name Wildcard [**] 165.247.107.139:137 ->
MY.NET.24.34:137
05/01-11:31:01.381487  [**] SMB Name Wildcard [**] 148.220.29.18:1033 ->
MY.NET.56.28:137
05/01-11:20:22.793455  [**] SMB Name Wildcard [**] 67.31.249.251:137 ->
MY.NET.24.34:137
05/01-11:20:23.161837  [**] SMB Name Wildcard [**] 61.141.87.194:1026 ->
MY.NET.31.99:137
```

To add to the above data, there was a set of 5 external source IP addresses that generated quite a bit of this traffic. Most of the source addresses appeared in anywhere from 1-100 packets. The following hosts appeared in many more than that:

Count	Address
8410	133.82.241.150
2639	216.78.180.128
2031	195.167.225.233
1898	143.248.115.88
1503	66.1.191.80

For correlation purposes, I searched the alert logs for other instances from these hosts. There were a few records from the 66.1.191.80 source:

```
05/02-10:43:15.917571  [**] EXPLOIT x86 NOOP [**] 66.1.191.80:2208 ->
MY.NET.190.93:139
05/02-10:43:18.000293  [**] EXPLOIT x86 NOOP [**] 66.1.191.80:2218 ->
MY.NET.190.93:139
```

Unfortunately, this signature is one that is known to have many false positives. According to the Snort signature database, simply transferring large files can trigger it.

Checking into the scan logs, we find that top talker for this alert, the external host 133.82.241.150, was involved in a rather large scan of UDP port 137. It scanned 961 hosts on the internal network in a matter of about 15 minutes. The scanned hosts started with MY.NET.104.X and went all the way to MY.NET.252.X. The address was being changed sequentially in the 3rd octet and every number was hit. The 4th octet only had a handful numbers used (around 20 in most cases) with this scan. There were no entries in the OOS file with regards to this external host.

It would seem that along with spoofed addresses, some real addresses are being used for these probes. I checked a route server from one of a list that can be found at www.traceroute.org for all 5 of these networks. They are all actively routed on the Internet.

Conclusion and Recommendations:
In many circumstances these alerts can be false positives. They can be generated by legitimate NetBIOS name query traffic on a network. The data in the alerts that we have does not suggest that. These packets are externally sourced destined for out internal network on NetBIOS ports. Most of the source addresses may be spoofed. But, we are possibly seeing some real addresses mixed in with the spoofed sources. The malicious host(s) are possibly attempting to blend themselves in with the "noise" in an attempt to avoid being noticed. It is an IDS evasion technique that is often used, drown the IDS in meaningless logs and send your real packets through with them hoping to blend in with the garbage and get thrown out without detection. The final thought on this trace is that NetBIOS traffic should always be filtered at the borders, both ingress and egress. I would recommend that the network administrators immediately place access control lists at their borders to block NetBIOS traffic from getting in or out of their enterprise. We'll gather some additional information regarding the top talking external host later in this document.

spp_http_decode: IIS Unicode attack
Background:
This is another alert, as seen with the fragmentation alert above, which comes from a preprocessor. This seems to be a source of many false positives. The Snort-users archives are littered with people complaining about this particular preprocessor and false positives. In fact, in the Snort FAQ, (http://www.snort.org/docs/faq.html#4.17) there is a specific section with regards to this. The job of this preprocessor is to convert any Unicode to plain ASCII text so that it can be passed on the normal Snort rules and processed. Unicode is often used to disguise malicious web requests in order to compromise a system. This conversion helps Snort in dealing with web requests and determining if the activity is malicious or not. John Berkers, on the Snort-users mailing list archives, has a post in which he describes this process and some details about the preprocessor (http://archives.neohapsis.com/archives/snort/2001-08/0075.html). Apparently, sites with multi-byte characters tend to trigger this event (such as Simplified Chinese). Here is one of the CERT references with regards to directory traversal with the use of Unicode,

http://www.kb.cert.org/vuls/id/111677. Also, here is one of the CVEs with regards to directory traversal, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884.

Analysis:
This preprocessor rule generated 30,441 alerts. Below is a sample of the records found in the alerts file with regards to this alert.

```
05/01-11:20:11.136064  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.153.124:2274 -> 211.117.63.210:80
05/01-11:20:11.136064  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.153.124:2274 -> 211.117.63.210:80
05/01-11:20:11.136064  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.153.124:2274 -> 211.117.63.210:80
```

Also, the following are the top 5 talkers for this alert:

```
Count  Top 5 Sources
2388   MY.NET.153.143
2200   MY.NET.97.213
1825   MY.NET.153.176
1701   MY.NET.153.165
1321   MY.NET.153.149


Count  Top 5 Destinations
2482   218.153.6.197
2162   211.233.29.9
1997   218.153.6.229
1573   210.219.197.11
1483   218.153.6.244

Count  Top 5 Flows (Src/Dst Pairs)
2066   MY.NET.97.213   -> 211.233.29.9
1242   MY.NET.153.143  -> 218.153.6.229
887    MY.NET.153.143  -> 218.153.6.212
853    MY.NET.106.107  -> 210.219.197.11
839    MY.NET.97.216   -> 211.233.29.51
```

There is a lack of information with regards to these connections. It's difficult to know if these are false positives without seeing the actual packets. There has been quite a bit of chatter on mailing lists with regards to this rule triggering many false positives and filling up log files. A whois query indicates that all the top talking external hosts are parts of address blocks out of Korea.

Conclusion and Recommendations:
There is not enough information to draw to any reliable conclusions. It is well documented that cookies with URL encoded binary data can cause this. Certain types of pop-up ads have been known to trip of this. We would need the packet dumps to make a better assessment of what really is going on here.

Speaking of just the top talkers, it is very possible that these Korean use multi-byte characters on their corresponding web pages and that is what is triggering the alerts. Also, this looks to be

outbound traffic destined for port 80, which would support that theory as well. There is no inbound traffic, at least with the top talkers group.

From the shear number of sources and destination (804 sources and 880 destinations), I would definitely recommend more investigation into this. If this is not a false positive, the ramification can be terrible. Nimda, Code Red, Sadmind are nothing to scoff at. I would recommend further investigation and possible tuning of the preprocessor. One recommendation on tuning this came from Marty Roesch in a message on Snort-users (http://archives.neohapsis.com/archives/sf/ids/2001-q2/0286.html). He recommended a BPF filter at startup, something along the lines of:

```
snort <options> not port 80 and not net MY.NET
```

High port 65535 udp - possible Red Worm – traffic

Background:

Port 65535 is not an oft-used port. It is the last available TCP or UDP port. In this case, we have a bunch of alerts sourced and destined to port 65535 bi-directionally in and out of our network. I was not able to find much detail as to what this port is used for specifically. There are multiple Trojans listed for this port, but they are all TCP (which is covered later in this document). This seems to be local rules.

Analysis:

The first thing I did was gather some numbers on these packets. We have a total of 27,278 UDP alerts. I will look at UDP first. The first table (Table 3a) below lists out the top sources and destinations for the UDP traffic from these alerts, along with the number of destinations and sources they communicated with. The second table (Table 3b) lists the top source-destination port pairs. Those are followed by a few sample alerts.

Count	Top UDP Src	# of Dsts
13433	MY.NET.201.58	46
1839	65.120.111.17	1
1469	64.118.111.251	1
1045	66.42.68.210	1
945	62.75.136.123	1

Count	Top UDP Dst	# of Srcs
10628	MY.NET.201.58	27
1992	65.120.111.17	1
1678	66.42.68.210	1
1604	64.118.111.251	1
1114	12.235.90.8	1

Table 3a

Count	Top UDP Src Port – Dst Port Pairs
12959	65535 -> 5121
10373	5121 -> 65535
1495	6257 -> 65535
1282	65535 -> 6257

Table 3b

```
05/01-13:24:51.897144  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.201.58:65535 -> 68.96.49.118:5121
05/01-13:24:51.897155  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.201.58:65535 -> 24.165.6.237:5121
```

We can see from this data that a single host (MY.NET.201.58) on the internal network is responsible for 24,061 (88.2%) of these alerts (13,433 as a source and another 10,628 as a

destination). I did a quick search of the alerts file for this host along with the its most used source and destination ports in these connections:

Count	Src Port	Count	Dst Port
13421	65535	12952	5121
10370	5121	10629	65535

From this table we can see that almost all the communication from or to MY.NET.201.58 had a source or destination port of either 5121 or 65535. UDP port 5121 is the default port for a online game called Neverwinter Nights (http://nwn.bioware.com/support/techfaq.html). This is a multiplayer online game according to the following blurb from the official website:

> ...But the Neverwinter experience is not just for one person- adventure with all your friends. Neverwinter Nights can be played online with up to 64 friends, all sharing in the adventure...

The technical FAQ implies that connections must be permitted "through the firewall" for UDP 5121. My guess would be that the hosts involved in playing together would have to communicate with one another via this port. Another note to add to this, there was a total of 43 addresses that our internal gaming host was communicating with, supporting the idea that this gaming traffic.

Also, for correlation, I searched the scan logs for this particular host. The results also supported the notion that this is merely gaming traffic. We saw a total of 2166 UDP scans records involving this host. Of those, 1483 were destined for UDP either port 5121 (832 of those sourced from port 65,535) or one of the other recommended ports for Neverwinter Nights (5120-5129). The rest of the traffic was UDP packets sourced and destined to port 13139. This is the GameSpy (service used for online game play) custom UDP ping port.

This left us with the last two entries in the source-destination port pairs table (Table 3b) above:

```
1495    6257  -> 65535
1282    65535 -> 6257
```

UDP port 6257 is the default port WinMX, a peer to peer file sharing tool. This traffic may be just that, but we would need more information (for example a packet capture).

Conclusion and Recommendations:
In my estimation, the majority of the "UDP High Port" alerts are caused by a single host (MY.NET.201.58) playing an multiplayer online game across the Internet. All the evidence points to these alerts being mostly false positives. The other heavy hitters, even though the numbers weren't really comparable, seemed to be WinMX traffic. Other than possibly breaking copyright violations and Acceptable Use Policies, there is nothing malicious about that program. I would recommend that this rule be tuned to cut down on the noise. It seems to be too general, which makes it a poor and noisy rule.

CS WEBSERVER - external web traffic
Background:

This alert seems to be firing on every connection to this web server. The alerts are rather straight forward and not out of the ordinary. It looks as though this system provides web services to the world, but it seems a custom rule has been written to alert on that traffic when connections are made.

Analysis:
In the 5 day period, we saw 5326 unique source hosts attempting to make a connection to this web server via port 80. There were a total of 25,730 alerts generated by different packets from the above-mentioned 5326 sources. One interesting thing of note is that the majority of the connections coming into this server were from 1 IP address. 14,014 alerts out of the 25,730 (54.4%) were from the source host 216.39.48.127. This would be an interesting host to follow up do some investigating.

I checked the OOS and the scan files and did find some correlations. It seems that handful (89) of these packets had the ECN bits set along with the SYN bit. These may actually be normal IP packets, but they may be coming from segments on the Internet that utilize the ECN bits. ECN stands for Explicit Congestion Notification. It is a relatively new development that is meant to help with relieving congestion from heavily loaded links. ECN uses the two high order bits of the TCP flag byte. These bits used to be reserved. Part of the problem with ECN is that not all vendors have fully supported it and some will explicitly drop packets that have the ECN bits set. They consider them anomalous packets because the once reserved high order bits are set. For correlation purposes I looked through the scan file for this web server and found a bit data that went along with what was found in the OOS file.

Conclusion:
This seems to be alert that is specifically looking to log all web traffic destined for the MY.NET.100.165 web server. If that is the goal, then it is doing a fine job of it. If that wasn't the goal and this rule was put to monitor some malicious traffic, this rule would have been changed. At this point it is alerting on normal traffic. It would likewise alert on abnormal traffic, but because of the noise from the normal traffic, the malicious packets may go undetected. I would recommend making this rule something more useful than logging every HTTP connection.

See Link Graph below for these alerts.

High port 65535 tcp - possible Red Worm - traffic
Background:
TCP port 65535 has a handful of Trojans that are associated with it. There is no legitimate use for this port that we know of as yet. Some of the Trojans that are associated with it are:

```
65535 tcp   [trojan] Adore worm
65535 tcp   [trojan] RC1 trojan
65535 tcp   [trojan] Sins
```

There is a good description of the Adore worm and how it works here at DialogueScience, Inc. (http://www.dials.ru/english/inf/linux_adore.htm). It is a Linux worm propagates itself

previously known vulnerabilities in Washington University's ftp server (wu-ftpd), Remote Procedure Call stat server (rpc.statd), LPRng (lpd), and BIND DNS server (bind). According to the article, it uses the same methods as previous worms such as Linux.Ramen and Linux.Lion. The end result is a host that is listening on port 65535 with a root shell session waiting for a connection.

Analysis:
We found 23,641 TCP High Port alerts. We would like to do the same type of breakdown for TCP as we did for the UDP version of this alert. The first table below (Table 5a) is a list of the top sources and top destinations, along with the number of destinations and sources they communicated with. The second table (Table 4b) lists the top source-destination port pairs. Those are followed by a few sample alerts.

Count	Top TCP Src	# of Dsts
3945	130.85.201.38	1
3456	130.85.226.250	1
3295	67.161.246.193	1
2550	218.141.54.99	1
1697	213.161.3.60	1
1320	130.85.226.206	1
1215	217.127.167.6	1

Count	Top TCP Dst	# of Srcs
3944	67.161.246.193	1
3454	218.141.54.99	1
3294	130.85.201.38	1
2549	130.85.226.250	1
1697	130.85.226.206	1
1320	213.161.3.60	1
1212	130.85.233.134	1

Table 4a

Count	Top TCP Src Port - Dst Port Pairs
3944	4606 -> 65535
3454	1857 -> 65535
3293	65535 -> 4606
2549	65535 -> 1857
1697	65535 -> 4688
1320	4688 -> 65535
1212	65535 -> 1327
846	1327 -> 65535

Table 4b

```
05/01-17:24:21.277691  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.205.14:3933 -> 81.86.75.175:65535
05/01-17:24:25.207093  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.205.14:3933 -> 81.86.75.175:65535
```

There is definitely some correlation between the source-destination port pair and the top source and destination hosts. The alert numbers are almost identical (and in some cases they are). We can also see that all the top source hosts and top destination hosts are only communicating with one system.

The problem I had with analyzing this alert was a complete lack of information in every aspect. There were no entries in the OOS files and there were also no entries in the scan files. Port 65535 does not seem to be a destination on the internal network, which lowered my level of concern a bit. That said, only a two of the ports being used (other than 65535) are listed as having a legitimate use.

```
1857  tcp   DataCaptor
1327  tcp   Ultrex
```

I don't think either of these applies in this scenario. It is possible that this is some sort of custom application or something of that nature. But, looking at the external hosts, they all have broadband customer DNS names, for example:

```
Name:        c-67-161-246-196.client.comcast.net
Address:     67.161.246.196
```

Conclusion:
This would definitely need to be investigated further. The internal hosts would need to be examined to determine the source of this data and to determine if there has been a compromise. Without any packet captures or additional data, it is difficult to analyze this further. My suggestion would be to research the cause of this rule.

Scans

The scan log was processed next using a combination of custom scripts and Snortsnarf (www.silicondefense.com/software/snortsnarf/index.htm) [Scan Scripts]

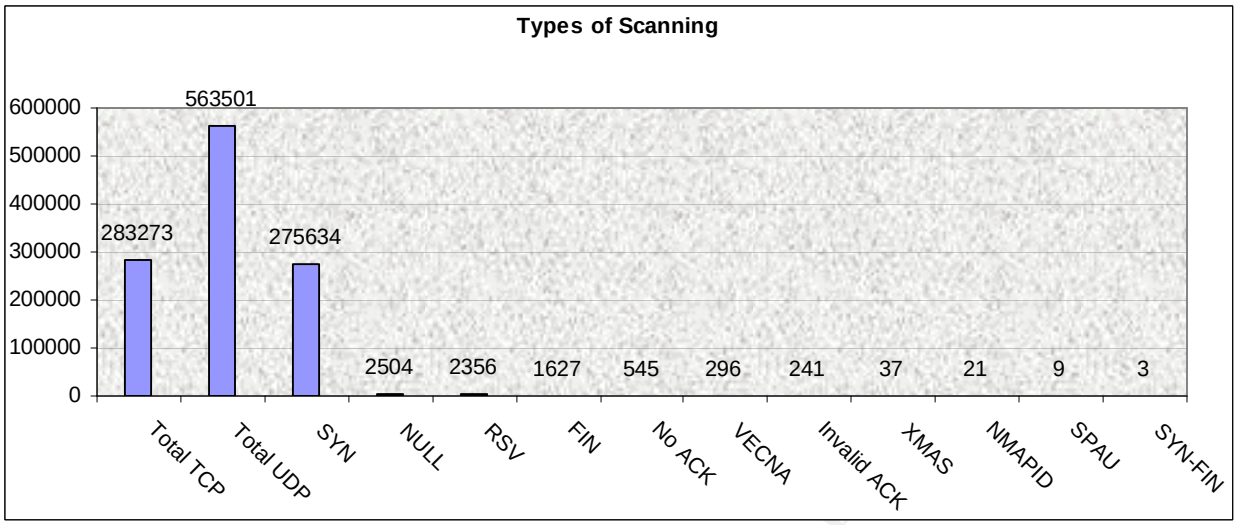
During the five-day period, a variety of scan types were logged. The first table (Table 3) and graphs (Graph 3 and Graph 4) provide details about TCP flag byte settings in each of the scans in the scan log for TCP and a general UDP line item.

TCP Port Scans Broken Out	
283273	Total Scans
275634	SYN Scans
2504	NULL Scans
2356	Reserved Bit Scans
1627	FIN Scans
545	No ACK Scans
296	VECNA Scans
241	Invalid ACK Scans
37	XMAS Scans
21	NMAPID Scans
9	SPAU Scans
3	SYN-FIN Scans

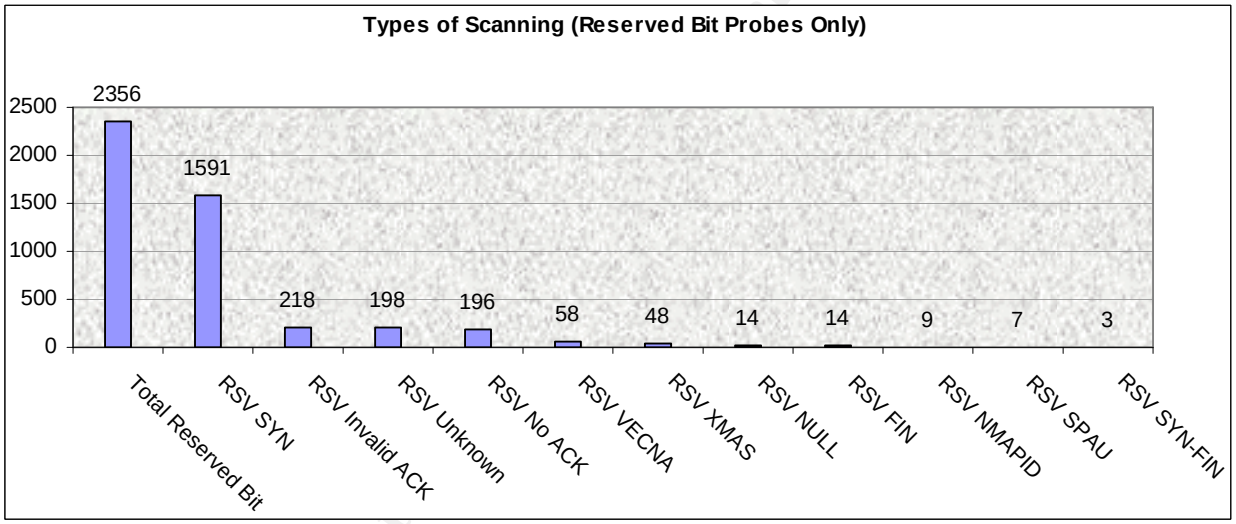
TCP Reserved (RSV) Bit Scans Broken Out	
2356	Total Reserved Bit Scans
1591	RSV SYN Scan
218	RSV Invalid ACK Scan
198	RSV Unknown Scan
196	RSV No ACK Scan
58	RSV VECNA Scan
48	RSV XMAS Scan
14	RSV NULL Scan
14	RSV FIN Scan
9	RSV NMAPID Scan
7	RSV SPAU Scan
3	RSV SYN-FIN Scan

Table 5

563501	Total UDP



Graph 3



Graph 4

We now have a good grasp what general types of scanning was going on. We wanted a better idea of what was being scanned, who was being scanned and who was doing the scanning. For this we analyzed the scan files for top source host scanners, top destination host scanners, top source ports and destination ports (all by the number of scans). (Table 6a and 6b).

Count	Top UDP Src IP
64658	130.85.210.114
39633	130.85.240.62
32605	130.85.87.50
29283	130.85.250.98
26384	130.85.97.190
21849	130.85.1.3
20866	130.85.234.158
16736	130.85.205.150

Count	Top UDP Dst IP
64602	213.97.198.23
1779	64.39.186.133
1737	66.66.126.241
1624	66.167.144.245
1620	24.42.0.66
1570	68.165.25.243
1219	68.13.93.150
1212	12.245.31.155

15285	130.85.153.152
13679	130.85.225.230

1186	68.81.50.22
1179	68.82.22.172

Count	Top UDP Src Port
43924	6257 WinMX
32417	27022
29443	2921
24546	7674
21822	32832
20877	2315
16867	2468
16756	3708
15554	1025
13692	2305

Count	Top UDP Dst Port
77862	137 NB NS
41738	6257 WinMX
25287	53 DNS
24547	7674
16088	27005 FlexLM
13592	22321 Wnn6
11260	0 Unknown
6055	1214 Kazaa
5949	13139 GameSpy
5059	43620

Table 6a

Count	Top TCP Src IP
15962	152.1.193.6
13949	217.88.231.137
11688	217.84.122.16
10633	130.85.97.181
9160	198.144.65.56
8313	64.212.144.139
8244	80.161.34.13
7663	66.130.208.97
7037	213.204.66.141
6939	208.163.46.185

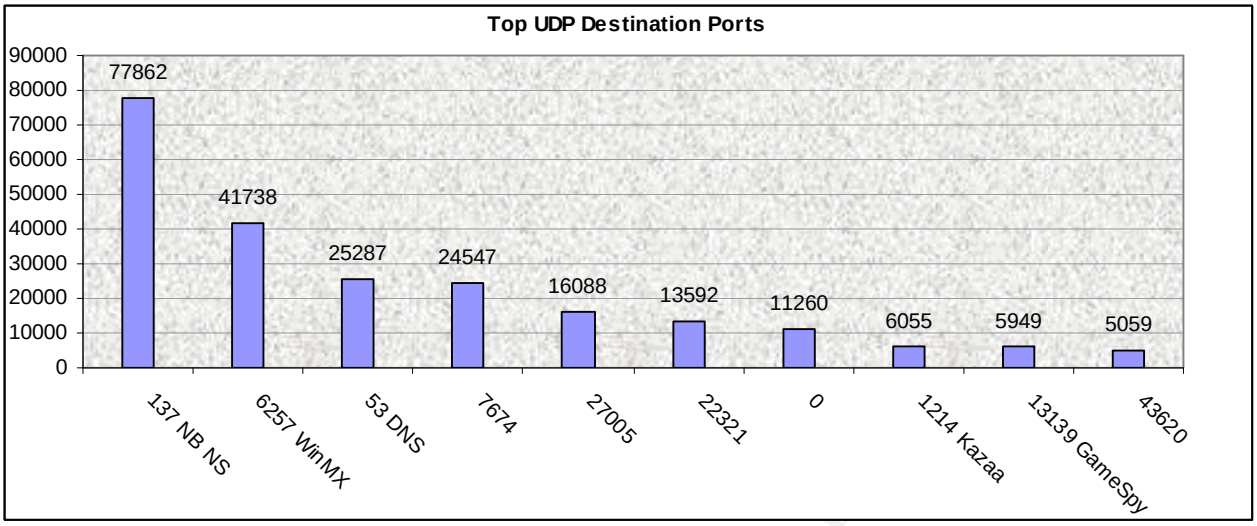
Count	Top TCP Dst IP
15967	130.85.132.26
924	130.85.234.82
457	130.85.249.194
310	130.85.238.230
285	130.85.207.254
235	130.85.86.66
213	130.85.218.254
206	130.85.6.7
175	200.77.81.95
145	130.85.211.26

Count	Top TCP Src Port
3819	139 NB SSN
2932	0 Unknown
152	1131
147	1128
143	1130
143	1125
143	1120
138	1107
138	1103
137	1142

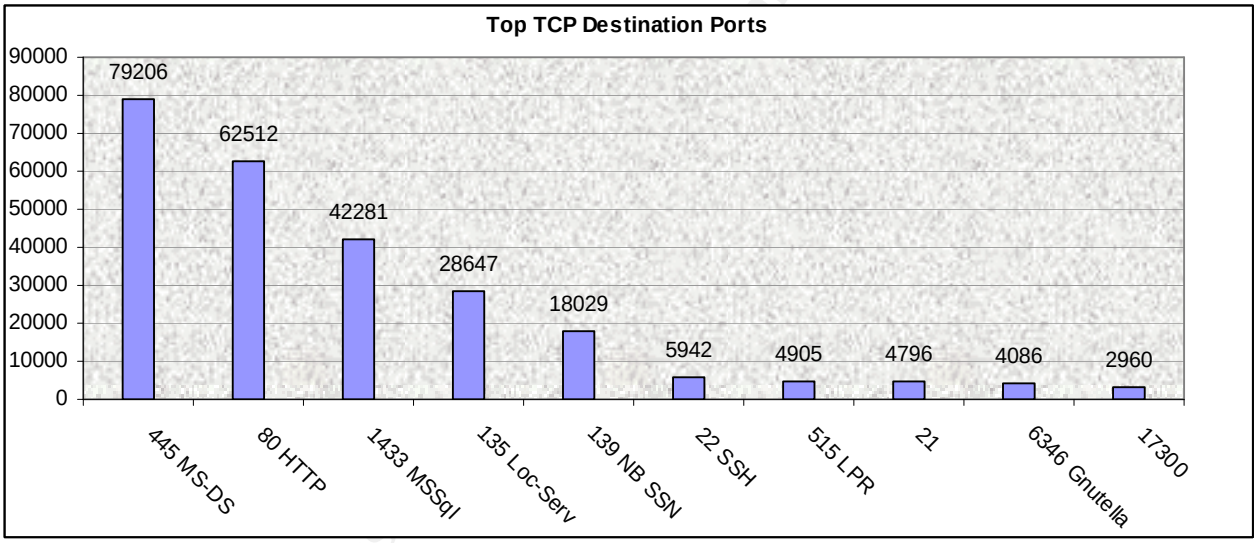
Count	Top TCP Dst Port
79206	445 MS-DS
62512	80 HTTP
42281	1433 MSSql
28647	135 Loc-Serv
18029	139 NB SSN
5942	22 SSH
4905	515 LPR
4796	21 FTP
4086	6346 Gnutella
2960	17300 Kuang2 Virus

Table 6b

The following graphs will help visualize this numbers (Graph 4a and 4b).



Graph 4a



Graph 4b

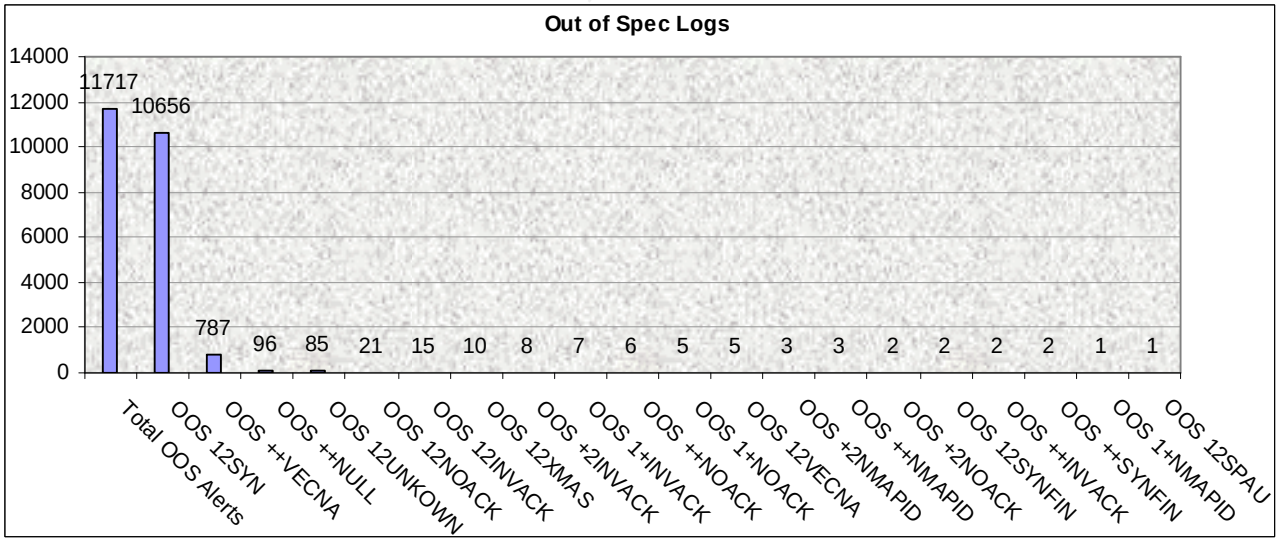
Out Of Spec

The out of spec file contains records of packets that were somehow not quite right. For example, it contained the packets that had the reserved bit set in the TCP flag byte. Another example would be all the packets that had a bad combination of the TCP flags set (i.e. NULL, XMAS, SYN/FIN). Below is the table (Table 7) and breakout of how many of each type was seen. In the table and the graph that follows, the first two characters of each type mark the reserved bits. A 12SYN, for example, indicates SYN packets with both the ECN bits set. In a case where no ECN bit is set, a plus sign "+" indicates that. For example ++VECNA indicates a VECNA scan with no reserved bits set. 1+VECNA would indicate a VECNA scan with only the first ECN bit set.

Count	Type of OOS Alert
11717	Total OOS Alerts
10656	OOS 12SYN
787	OOS ++VECNA
96	OOS ++NULL
85	OOS 12UNKOWN
21	OOS 12NOACK
15	OOS 12INVACK
10	OOS 12XMAS
8	OOS +2INVACK
7	OOS 1+INVACK
6	OOS ++NOACK
5	OOS 1+NOACK
5	OOS 12VECNA
3	OOS +2NMAPID
3	OOS ++NMAPID
2	OOS +2NOACK
2	OOS 12SYNFIN
2	OOS ++INVACK
2	OOS ++SYNFIN
1	OOS 1+NMAPID
1	OOS 12SPAU

Table 7

To help visualize, the table (Table 7) is graphed below (Graph 5).



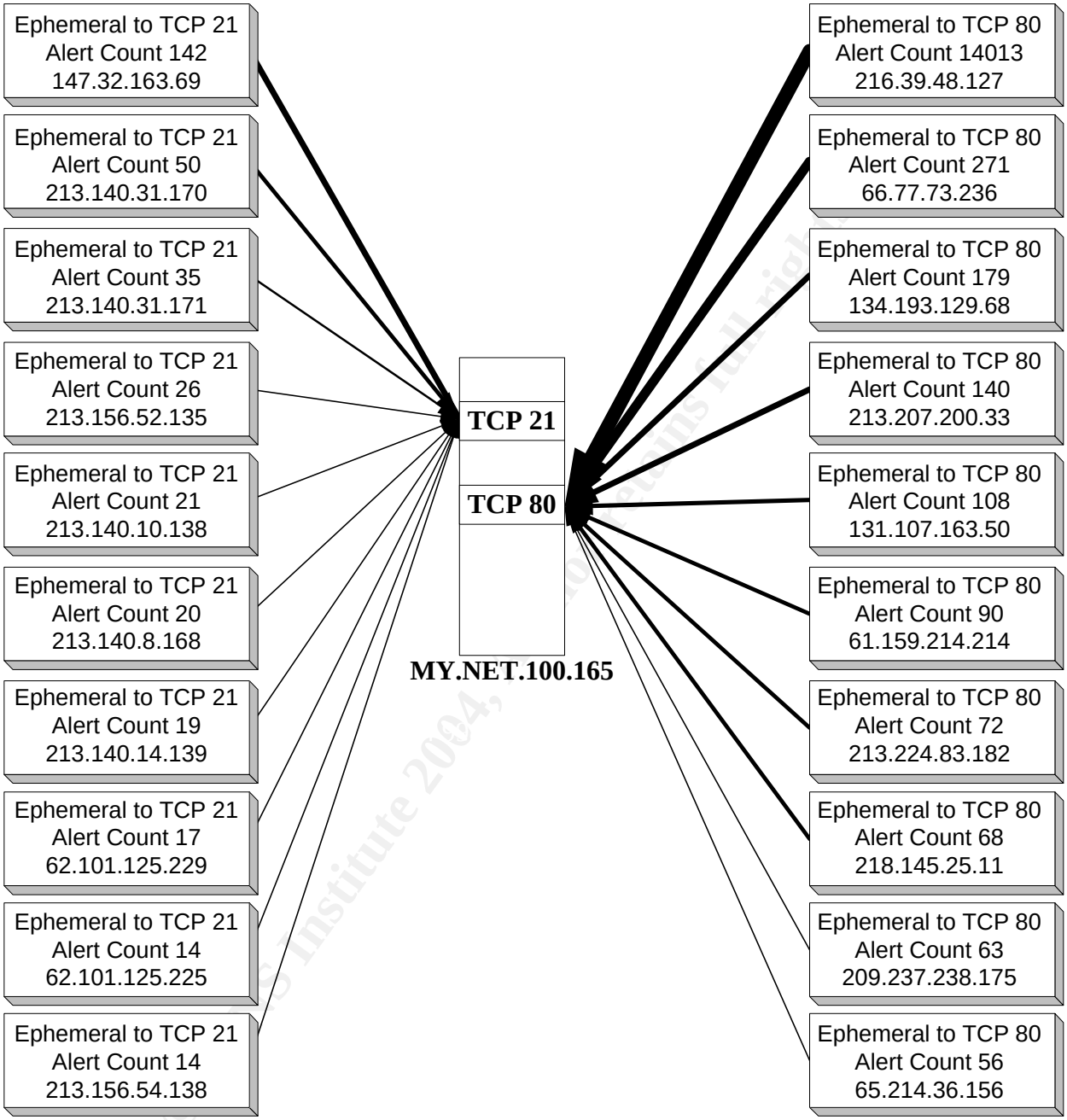
Graph 5

From the table and graph, we see that SYN packets with the reserved bit set are the most popular entry. This could be a viewed in a variety of ways. It could have come from a network on the Internet that is utilizing the ECN bits. It could be that someone is attempting to the ECN bits as some sort of stealth technique. To get a better idea, we will look at the talkers for 12SYN.

Count	Top Src IP
1562	209.123.49.137
1318	68.54.93.181
462	213.197.10.95
365	81.218.114.59
338	64.28.101.9
318	210.233.23.128
270	81.218.109.79
250	66.140.25.157
201	210.253.214.117
129	151.42.126.19
117	216.95.201.33
104	193.233.7.104

It appears that 209.123.49.137 and 68.54.93.181 are the top talkers with regards to SYN packets with the ECN bits set. We will gather some more information regarding the two top talkers later in this document.

LINK GRAPH



INTERESTING EXTERNAL HOSTS

Top Talker of Incomplete Fragment Alert 213.97.198.23	**Top Talkers with SYN Packets with ECN Bits set 209.123.49.137**
inetnum: 213.97.0.0 - 213.97.255.255 netname: RIMA descr: Telefonica De Espana SAU (NCC#2000013794) descr: Red de servicios IP descr: Spain country: ES admin-c: LJP5-RIPE tech-c: FLT14-RIPE rev-srv: scmrro3.nombres.ttd.es rev-srv: scmrro4.nombres.ttd.es rev-srv: ns.ripe.net status: ASSIGNED PA remarks: ********** remarks: For ABUSE/SPAM/INTRUSION issues remarks: PLEASE CONTACT THROUGH LINK remarks: http://www.telefonicaonline.com/nemesys/ remarks: or send mail to nemesys@telefonica.es remarks: any mail to adminis.ripe@telefonica.es will be ignored remarks: ********** notify: adminis.ripe@telefonica.es mnt-by: MAINT-AS3352 changed: adminis.ripe@telefonica.es 20000302 changed: adminis.ripe@telefonica.es 20020530 changed: administracion.ripe@telefonica-data.com 20030121 source: RIPE route: 213.97.0.0/16 descr: TTDNET (Red de servicios IP) origin: AS3352 mnt-by: MAINT-AS3352 mnt-routes: MAINT-AS3352 mnt-lower: MAINT-AS3352 changed: administracion.ripe@telefonica-data.com 20010308 changed: administracion.ripe@telefonica-data.com 20020118 changed: administracion.ripe@telefonica-data.com 20020313 source: RIPE	OrgName: Net Access Corporation OrgID: NAC Address: 1719 STE RT 10E Address: Suite 111 City: Parsippany StateProv: NJ PostalCode: 07054 Country: US NetRange: 209.123.0.0 - 209.123.255.255 CIDR: 209.123.0.0/16 NetName: NAC-NETBLK02 NetHandle: NET-209-123-0-0-1 Parent: NET-209-0-0-0-0 NetType: Direct Allocation NameServer: NS1.NAC.NET NameServer: NS2.NAC.NET NameServer: NS5.NAC.NET Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE Comment: Comment: * Reassignment information for this network is available Comment: * available at whois.nac.net 43 RegDate: 1997-08-06 Updated: 2001-08-22 TechHandle: ZN77-ARIN TechName: Net Access Corporation TechPhone: +1-800-638-6336 TechEmail: legal@nac.net 68.54.93.181 Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1) 68.32.0.0 - 68.63.255.255 Comcast Cable Communications, Inc. BALTIMORE-A-4 (NET-68-54-80-0-1) 68.54.80.0 - 68.54.95.255

Top Talkers of IIS Unicode Attack 211.233.29.9 and 211.233.29.51

IP Address : 211.233.28.0-211.233.31.255
Network Name : KIDC-INFRA-SERVERROOM-DAUM
Connect ISP Name : KIDC
Connect Date : 20001213
Registration Date : 20011115

[Organization Information]
Orgnization ID : ORG231919
Org Name : Daum Communication

State : SEOUL
Address : Gangnam-gu, Yeoksam-dong, DACOM B/D 12F. 706-1
Zip Code : 135-987

[Admin Contact Information]
Name : Hanju Kim
Org Name : Daum Communication
State : SEOUL
Address : Gangnam-gu, Yeoksam-dong, DACOM B/D 12F. 706-1
Zip Code : 135-987
Phone : +82-2-6446-6407
Fax : +82-2-6446-6499
E-Mail : hankim@daumcorp.com

[Technical Contact Information]
Name : youngchul Lee
Org Name : Daum Communication
State : SEOUL
Address : Gangnam-gu, Yeoksam-dong, DACOM B/D 12F. 706-1
Zip Code : 135-987
Phone : +82-2-6446-6407
Fax : +82-2-6446-6499
E-Mail : uniace@daumcorp.com

Top Talker of SMB Wildcard Alert 133.82.241.150

Network Information:
a. [Network Number] 133.82.0.0
b. [Network Name] CU-NET
g. [Organization] Chiba University
m. [Administrative Contact] SS1986JP
n. [Technical Contact] SO014JP
n. [Technical Contact] YN3644JP
n. [Technical Contact] MO4342JP
p. [Nameserver] nanohana.cix.chiba-u.ac.jp
p. [Nameserver] ns.chiba-u.ac.jp
y. [Reply Mail] cunet-admin@chiba-u.ac.jp
[Assigned Date]
[Return Date]
[Last Update] 2002/04/12 11:15:36 (JST) okano@imit.chiba-u.ac.jp

Top Talkers of IIS Unicode Attack 218.153.6.229 and 218.153.6.212

IP Address : 218.153.4.0-218.153.11.255
Network Name : KORNET-NETINFRA-JUNGANG
Connect ISP Name : KORNET
Connect Date : 20020812
Registration Date : 20030516

[Organization Information]
Orgnization ID : ORG204104
Org Name : CENTRAL DATA COMMUNICATION OFFICE
State : SEOUL
Address : 128-9 YEUNKEONDONG JONGROKU
Zip Code : 110-460

[Admin Contact Information]
Name : DongJoo Lee
Org Name : KOREA TELECOM
State : SEOUL
Address : 128-9 Youngundong Chongroku

Zip Code : 110-460
Phone : +82-2-747-9213
Fax : +82-2-747-8701
E-Mail : ip@ns.kornet.net

[Technical Contact Information]
Name : GyungJun Kim
Org Name : KOREA TELECOM
State : SEOUL
Address : 128-9 Youngundong Chongroku

Zip Code : 110-460
Phone : +82-2-747-9213
Fax : +82-2-747-8701
E-Mail : ip@ns.kornet.net

210.219.197.11

[ISP IP Admin Contact Information]
Name : IP Administrator
Phone : +82-2-3149-4999
Fax : +82-2-365-4046
E-Mail : ip-adm@elim.net

[ISP IP Tech Contact Information]
Name : IP manager
Phone : +82-2-3149-4999
Fax : +82-2-365-4046
E-Mail : ip@elim.net

[ISP Network Abuse Contact Information]
Name : Network Abuse
Phone : +82-2-3149-4999
Fax : +82-2-365-4046
E-Mail : abuse@elim.net

OOS Scripts

reformat-oos

```
#!/usr/bin/perl -w
# General Plan:
# Skip lines till we find a line that begins with \d\d/\d\d-\d\d:
# (i.e., a date/time looking entity).
# Join lines together that are part of this entry into an single line.
# Special fiddling for entries that have a data portion, so we put all
# the hex together followed by all the ASCII, rather than intermingling
# them, as would happen if we just catenated the lines.
#
# All data is read from standard input, and the results are written
# to standard output.

use strict;

sub findstart();
sub processentry($);

# main
{
  while(my $line = findstart()) {
print "finstart returned ($.) = $line.\n";
    $line = processentry($line);
    print "$line\n";
  }
  exit(0);
} # main

sub findstart() {
  while(my $line = <>) {
    chomp($line);
    next if($line !~ /^\d\d\/\d\d-\d\d:/);
    return($line);
  }
  return(undef);
}

sub processentry($) {
  my($line) = @_;
  my($asc, $hex);

  while($line !~ /TcpLen:\s+\d+/) {
    my $curline = <>;
    chomp($curline);
    $line .= " " . $curline;
  }
# Now we have all the pieces of the header - sniff a bit to see
# if there is additional data - don't depend on length fields
# in the header, since I'm not real sure how they relate to the
# data as formatted.  It appears that DgmLen values > 60 imply
# DgmLen - 60 + TcpLen data bytes follow, but...
  $asc = "";
  while(my $curline = <>) {
    chomp($curline);
    if($curline =~ /^TCP Options/) {
      $line .= " " . $curline;
      next;
    }
    return($line . " " . $asc) if($curline =~ /^\s*$/);
    $line .= " " . substr($curline,0,47);
```

```
    $line =~ s/\s+$/ /;
    $asc .= substr($curline,49);
  }
  return($line . " " . $asc);
}
#
#-end
```

Scan Scripts

tcp-scantype

```
#!/bin/tcsh
#
# Each step is annotated below

# Break the all.scan file (all 5 days) into UDP/TCP
#
grep UDP all.scans > all.UDP.scans
grep -v UDP all.scans > all.TCP.scans

# SYN Scans, only SYN flag set
#
grep "\*\*\*\*\*\*\*S\*" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.SYN.scans

# NULL Scans, no TCP flags set
#
grep "NULL" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.NULL.scans

# FIN Scans, only FIN flag set
#
grep "\*\*\*\*\*\*\*\*F" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.FIN.scans

# Scans, combination of U,P,R,S,F without an ACK
# Some flag combinations (may include reserved bits)
# **U***S*
# **U***SF
# **U**R** , *****R*F
# **U**R*F , *****RS*
# **U**RS* , *****RSF
# **U**RSF , ****P*S*
# **U*P*S* , ****P*SF
# **U*PR** , ****PR**
# **U*PR*F , ****PR*F
# **U*PRS* , ****PRS*
# **U*PRSF , ****PRSF
#
grep "NOACK" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.NOACK.scans

# Incomplete XMAS scan with
# http://lists.insecure.org/lists/nmap-hackers/1999/Oct-Dec/0012.html
# Some flag combinations (may include reserved bits)
# **U*****
# ****P***
# **U*P***
# ****P**F
# **U****F
#
grep "VECNA" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.VECNA.scans
```

```
# Invalid ACK scans
# http://archives.neohapsis.com/archives/snort/2000-03/0241.html
# Some flag combinations (may include reserved bits)
# **UA**S* ,
# **UA**SF , ***A**SF
# **UA*R** , ***A*R*F
# **UA*R*F , ***A*RS*
# **UA*RS* , ***A*RSF
# **UA*RSF , ***AP*S*
# **UAP*SF , ***AP*SF
# **UAPR** , ***APR*F
# **UAPR*F , ***APRS*
# **UAPRS* , ***APRSF
#
grep "INVALIDACK" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.INVACK.scans


# XMAX scan, all bits set
#
grep "XMAS" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.XMAS.scans

# Part of Nmap OS fingerprinting scan
# http://archives.neohapsis.com/archives/snort/2000-02/0055.html
#
grep "NMAPID" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.NMAPID.scans

# SYN/FIN scans, only these flags set
# http://archives.neohapsis.com/archives/snort/2000-07/0180.html
#
grep "SYNFIN" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.SYNFIN.scans

# SPAU scans, SYN, PSH, ACK, and URG flags set
#
grep "SPAU" all.TCP.scans | grep -v "RESERVEDBITS" > all.TCP.SPAU.scans

# Reserved Bit Scans, one or both Reserved bits set
#
grep "RESERVEDBITS" all.TCP.scans > all.TCP.RSV.scans

   # Traffic with reserved bits broken out
   #
   grep "\*\*\*\*S\*" all.TCP.RSV.scans > all.TCP.RSV.SYN.scans
   grep "NULL" all.TCP.RSV.scans > all.TCP.RSV.NULL.scans
   grep "\*\*\*\*\*F" all.TCP.RSV.scans > all.TCP.RSV.FIN.scans
   grep "NOACK" all.TCP.RSV.scans > all.TCP.RSV.NOACK.scans
   grep "VECNA" all.TCP.RSV.scans > all.TCP.RSV.VECNA.scans
   grep "INVALIDACK" all.TCP.RSV.scans > all.TCP.RSV.INVACK.scans
   grep "XMAS" all.TCP.RSV.scans > all.TCP.RSV.XMAS.scans
   grep "NMAPID" all.TCP.RSV.scans > all.TCP.RSV.NMAPID.scans
   grep "SYNFIN" all.TCP.RSV.scans > all.TCP.RSV.SYNFIN.scans
   grep "SPAU" all.TCP.RSV.scans > all.TCP.RSV.SPAU.scans
   grep "UNKNOWN" all.TCP.RSV.scans > all.TCP.RSV.UNKNOWN.scans

# Reporting Functions
#
rm tcp-scantype-report
touch report-tcp
touch report-tcp-rsv
touch tcp-scantype-report
#
wc all.TCP.FIN.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.INVACK.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.NMAPID.scans | awk '{print $1,"\t"$4}' >> report-tcp
```

```
wc all.TCP.NOACK.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.NULL.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.SPAU.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.SYN.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.SYNFIN.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.VECNA.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.XMAS.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.scans | awk '{print $1,"\t"$4}' >> report-tcp
wc all.TCP.RSV.scans | awk '{print $1,"\t"$4}' >> report-tcp
#
wc all.TCP.RSV.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.FIN.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.INVACK.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.NMAPID.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.NOACK.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.NULL.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.SPAU.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.SYN.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.SYNFIN.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.UNKNOWN.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.VECNA.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv
wc all.TCP.RSV.XMAS.scans | awk '{print $1,"\t"$4}' >> report-tcp-rsv

cat report-tcp | sort -nr | sed 's/all\./  /g' | \
   sed 's/\./ /g' | sed 's/ TCP scans/--++ALL TCP Scans++--/g' | \
   sed '1s/^/\nTCP Port Scans Broken Out\n/g' | \
   sed '3s/^/------------------------\n/g' >> tcp-scantype-report

cat report-tcp-rsv | sed 's/^[ \t]*//'| sort -nr | \
   sed 's/all\./  /g' | sed 's/\./ /g' | \
   sed 's/ TCP RSV scans/--++ALL TCP RSV Scans++--/g' | \
   sed '1s/^/\nItems w\/TCP Reserved Bits --detail-- \n/g' | \
   sed '3s/^/-----------------------------------\n/g' >> tcp-scantype-report

rm report-tcp
rm report-tcp-rsv
```

tcp-dst-ip

(one of several, for each of "sort")

```
#!/bin/sh
#
rm tcp-dst-ip-report
touch tcp-dst-ip-report

FILES=" \
all.TCP.SYN.scans \
all.TCP.NULL.scans \
all.TCP.RSV.scans \
all.TCP.FIN.scans \
all.TCP.NOACK.scans \
all.TCP.VECNA.scans \
all.TCP.INVACK.scans \
all.TCP.XMAS.scans \
all.TCP.NMAPID.scans \
all.TCP.SPAU.scans \
all.TCP.SYNFIN.scans \
all.TCP.RSV.SYN.scans \
all.TCP.RSV.INVACK.scans \
all.TCP.RSV.UNKNOWN.scans \
all.TCP.RSV.NOACK.scans \
all.TCP.RSV.VECNA.scans \
all.TCP.RSV.XMAS.scans \
```

```
all.TCP.RSV.NULL.scans \
all.TCP.RSV.FIN.scans \
all.TCP.RSV.NMAPID.scans \
all.TCP.RSV.SPAU.scans \
all.TCP.RSV.SYNFIN.scans
"
#

for file in $FILES; do
echo "" >> tcp-dst-ip-report
echo "Top 20 Destination IP Addresses for$file" | sed 's/all//g' | sed 's/\./ /g' >>
tcp-dst-ip-report
echo "  Count IP-Address" >> tcp-dst-ip-report
echo "  ----- ----------" >> tcp-dst-ip-report
cat $file | awk '{print $6}' | awk -F: '{print $1}' | sort | uniq -c | sort -r | head
-20 >> tcp-dst-ip-report
# echo ""
done
```

Refrences:

Linux NFS Material
http://marc.theaimsgroup.com/?l=linux-nfs
https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=58084
http://www.faqs.org/rfcs/rfc1191.html

SANS Information and Data
www.incidents.org/logs/Raw/2002.10.15
www.incidents.org/logs/Raw/README

Snortsnarf
www.silicondefense.com/software/snortsnarf/index.htm

List of Route Servers, Proxies, Looking Glass
www.traceroute.org

IEEE OUI Table
http://standards.ieee.org/regauth/oui/oui.txt

Ports List
www.neohapsis.com/neolabs/neo-ports/neo-ports.html

p0f Passive OS Fingerprinting Tool (fingerprint table)
www.stearns.org/p0f/p0f.fp

DeepSight Analyzer by SecurityFocus
http://analyzer.securityfocus.com

Dshield
www.dshield.org

myNetWatchman
www.mynetwatchman.com

BGP Routing Table Archives
http://archive.routeviews.org

Ofir Arkin (ofir@atstake.com)
Subject: A crash course with Linux Kernel 2.4.x, IP ID values & RFC 791
http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1

Fyodor {fyodor@insecure.org)
Remote OS detection via TCP/IP Stack FingerPrinting
http://www.insecure.org/nmap/nmap-fingerprinting-article.html
[Forescout]. "The First 15 Minutes - Critical Technical Considerations for Defending Enterprise Networks Against the Next Wave of Internet Threats" - http://www.forescout.com

Harrington, Chad [Entercept]. "Defense in Depth: Combining Behavioral Rules and Signatures" - http://www.entercept.com/products/entercept/whitepapers/downloads/defenseindepth.pdf

DeShon, Markus [SecureWorks]. "Intrusion Prevention versus Intrusion Detection" - http://www.secureworks.net/techResourceCenter/fullTechArticle.php?article=IpsVsIds

[Netscreen]. "Intrusion Detection and Prevention - Protecting Your Network from Attacks" - www.netscreen.com

Cummings, Joanne. "From Intrusion Detection to Intrusion Prevention" Network World Fusion, Sept. 23, 2002. http://www.nwfusion.com/buzz/2002/intruder.html

Information Systems Security Association -- Austin Chapter. Incident Response Plan Template. http://austin.issa.org/WhitePapers/ISSAITD.pdf