Global Information Assurance Certification Paper

Intrusion Detection In Depth

GCIA Practical Assignment

Version 3.3

James Maher

SANS - Darling Harbour

February 3 - 8, 2003

Submitted: 16/07/2003

Table of Contents

Assignment 1 - State of Intrusion Detection

'A Brief History of TimeX ,

..... and other mapping techniques'

Abstract

In this paper I will look at the use of active reconnaissance to probe and map an organisation's network perimeter. There will be a brief introduction to outline the fundamental reasons for network reconnaissance followed by an analysis of three increasingly sophisticated probing techniques. Each of which is intended to demonstrate the practical application of network reconnaissance to discover information pertaining to perimeter layout, network security policy, and any possible trust relationships, all key pieces of information for a targeted attack. Along with the detailed examination of the theory behind each exploit I will demonstrate the probe in action. Using network traces and correlated log entries I will then investigate means of initially identifying such probes and hopefully some methods for foiling such attempts. Finally for each example I will identify some areas in which early identification of such probes could be used to the advantage of the analyst with the propagation of false information.

Perimeter mapping

As you sit here reading this the networks and hosts you are entrusted to protect are under a constant barrage of attack from a variety of sources. The attacks can be broken down roughly in to two groups, targeted and non-targeted.

In the latter a 'script kiddie' may be randomly attacking huge swathes of address space, using the latest tool downloaded from packetstorm. These tools require

little to no knowledge of the exploit nor the hosts that are its intended victims. Generally these forms of attack tools simply use brute force to attack as many hosts as possible, based on the statistical probability of success given enough attempts. The intended goal is to compromise as many hosts as possible, and the attacker has no interest in whom these hosts may belong too prior to the attack.

A targeted attack however requires the attacker to have some knowledge of the system that is being attacked. Either who the victim is, what the system is, or both. A targeted attack is usually therefore preceded by some form of information gathering on the part of the would be villain. An attacker may be looking for a particular type of system to attack, to exploit a known vulnerability such as a recently published Sendmail vulnerability[1]. For this they will try and probe for SMTP servers, to discover what SMTP agent is running and what version it is.

In another scenario the attacker may be targeting a particular organisation, YOUR organisation. They will not be targeting a specific attack so their reconnaissance will be to find out as much as possible about your network and how you defend it. How is your network configured? What is your security policy? Are there any trust relationships that can be exploited? Once they have this information they can formulate some means of breaching your defenses.

It is the security analyst's job in all scenarios to stop such attacks. With targeted attacks however we cannot only protect against the attack itself but also the leakage of security information that a would be hacker will find invaluable. In addition to this the network analyst must be able to spot possible signs of reconnaissance and use this information to their advantage.

1 Firewall spotting

In a simpler and friendlier Internet, a long long time ago security was very much an afterthought. A large number of hosts on the Internet were not protected by even a simple filtering router, let alone a stateful inspection engine or an application proxy.

So a fundamental question a hacker of such times may want answered was:

"Is the target I am attacking protected by a firewall?"

One method of finding out was by taking advantage of the implementation of many older packet-filtering firewalls[2]. Using shortcuts in the implementation of the TCP/IP stacks of such devices, some packet filter engines would allow the attacker to identify whether or not a firewall was blocking traffic rather than a service simply not being present on the targeted host.

This form of reconnaissance was based on tricking the firewall into responding to a packet that a normal IP stack would not. According to the RFC for TCP[3] a TCP packet should contain a valid checksum. Should a packet with an invalid

[1]CERT® Advisory CA-2003-07 Remote Buffer Overflow in Sendmail -March 03 2003
[2]Note: this is not true of many more modern stateful packet filtering firewalls
[3]RFC793

checksum arrive at a host a complying stack should ignore the packet. However a number of packet filtering engines employed in older firewalls did not check this checksum as it was considered an unnecessary overhead[4]. This could be exploited by a hacker.

First send a packet with a bad TCP checksum to a host that you think may be behind a simple packet filtering firewall. If you receive a reset then it is most likely from a packet filtering firewall that does not bother to check the TCP checksum. If you get no response then the packet is most likely getting to the end host and the stack is rejecting the packet due to the bad checksum. This technique is extremely basic and does not take into consideration a number of factors. No modern packet filtering engines that I have been able to find are at all susceptible to this attack as they check the TCP checksum[5]. Secondly many firewalls are configured to drop packets rather than sending a reset packet, therefore our assumption that the lack of a reset indicates a listening device would again be flawed. Finally there is the assumption that the host you are targeting does indeed have a compliant TCP stack.

However it is a simple demonstration of one category of IDS/Firewall evasion which can be used to generate some basic perimeter information, and was the precursor to the more advanced techniques discussed in this paper. It is a simple form of Insertion attack. This category of attack is one whereby the attacker crafts a packet that an IDS or firewall will accept but an end-system rejects. Usually this is done to fool the firewall or IDS into believing the host-attacker's connection is in a different state than it is, which may allow the attacker to send data to the host it otherwise might not.[6] In this demonstration we are interested purely in the behavior of the firewall itself. Is it susceptible to this form of insertion attack?

Acquiring and compiling hping2

This quick demonstration will utilise hping2 a command line tool, styled on the more ubiquitous ping program, with much greater versatility.[7] The hping2 utility can be downloaded as source from http://www.hping.org, or you can download it in binary format as a package for some Linux distributions such as Debian. I installed my client via 'apt-get'[8] from a Debian apt mirror, but also downloaded the source to allow for inspection.

[4] As recently as Mar 15 2001 this allegation was being leveled at the PIX firewall from Cisco (v5.3)
[5] As will be seen in my attempts to demonstrate this prototype perimeter mapping technique
[6] An excellent description of this form of IDS evasion can be found in "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" http://secinf.net/info/ids/idspaper/idspaper.html
[7] Hping "supports TCP, UDP, ICMP and RAW -IP protocols, has a traceroute mode, the ability to send files between a covered[sic] channel, and many other features. " - Sanfilippo, Salvatore
[8] 'apt-get' is a Debian package management system.

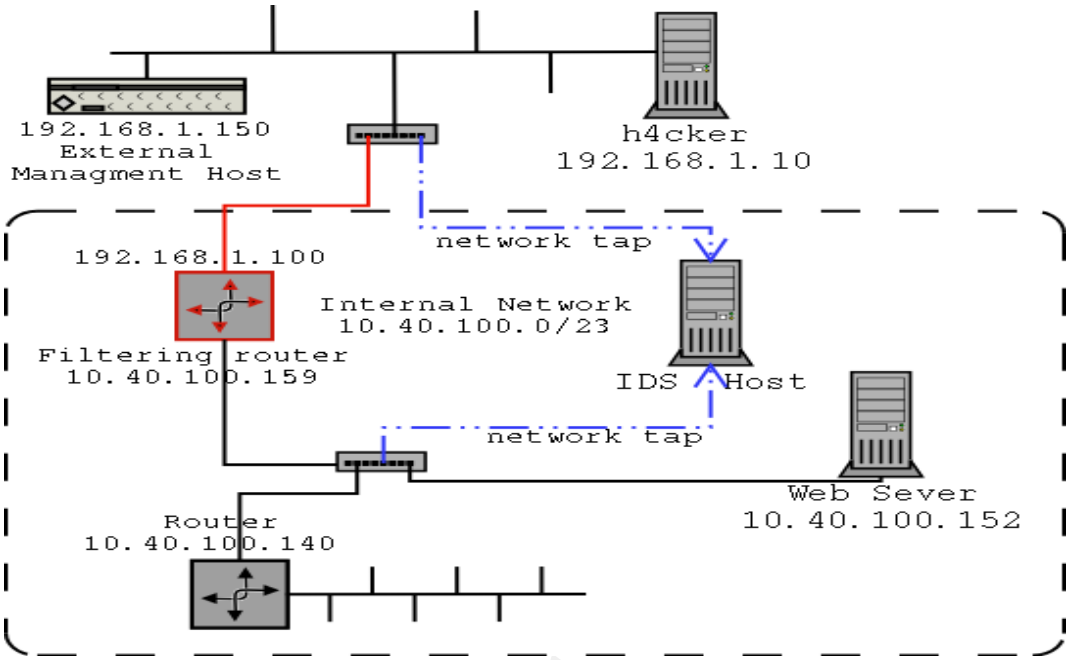


Fig 1 - Lab Network (part-1 & 2)

To exploit the TCP header vulnerability discussed earlier, I set up a simple lab network outlined above in fig1.

The firewall I used for the demonstration was iptables v1.2, which should not be fooled by the crafted packets.[9] It is configured to send RST packets or UDP Port unreachables rather than silently dropping packets. The 'hacker host', on the 192.168.1.0/24 external network is attempting to see if there is a firewall between this host and the targeted web server. The web server is visible from this external network, but only responds to pings, and HTTP requests.

The firewall rules implemented on the firewall are listed below:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  10.40.100.121       10.40.100.159       tcp dpt:22
ACCEPT     udp  --  10.34.100.1         10.40.100.159       udp spt:53
ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 8
ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 0
DROP       udp  --  0.0.0.0/0           0.0.0.0/0           udp dpts:137:139
LOG        all  --  0.0.0.0/0           0.0.0.0/0           LOG level 4 prefix `DROP IN'

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0           192.168.1.10        tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0           0.0.0.0/0           tcp spt:80 dpts:1025:65535
ACCEPT     tcp  --  0.0.0.0/0           0.0.0.0/0           tcp spt:53 dpts:1025:65535
ACCEPT     udp  --  0.0.0.0/0           0.0.0.0/0           udp dpt:53
ACCEPT     udp  --  0.0.0.0/0           0.0.0.0/0           udp spt:53
LOG        tcp  --  0.0.0.0/0           0.0.0.0/0           LOG level 4 prefix `DROP FORWARD tcp'
ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 8
ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 0
ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 11
LOG        udp  --  0.0.0.0/0           0.0.0.0/0           LOG level 4 prefix `DROP FORWARD udp'
```

[9] I was unable to find a current firewall which was susceptible to this attack but tho ught a demonstration and trace analysis was still useful.

```
REJECT     tcp  --  0.0.0.0/0        0.0.0.0/0        reject-with tcp-reset
REJECT     udp  --  0.0.0.0/0        0.0.0.0/0        reject-with icmp-port-unreachable

Chain OUTPUT (policy ACCEPT)
target     prot opt source           destination
LOG        all  --  0.0.0.0/0        10.0.0.0/8       LOG level 4 prefix `SPOOF ->Ext '
DROP       all  --  0.0.0.0/0        10.0.0.0/8
```

First I sent a TCP Syn packet to port 80 on the target host using the following hping command:

```
$ hping2 -V -p 80 -S 10.40.100.152
  using eth0, addr: 192.168.1.10, MTU: 1500
  HPING 10.40.100.152 (eth0 10.40.100.152): S set, 40 headers + 0 data bytes
  len=46 ip=10.40.100.152 flags=SA DF seq=0 ttl=63 id=0 win=5840 rtt=0.5 ms
  tos=0 iplen=44 seq=2998202135 ack=1207064129 sum=c90b urp=0
```

As we can see, we got a response presumably from the webserver. Lets check:

```
# tcpdump -ni eth0[10] 'ip'[11]
  05:09:52.927042 192.168.1.10.1211 > 10.40.100.152.80: S 1207064128:1207064128(0) win 512
  05:09:52.927155 10.40.100.152.80 > 192.168.1.10.1211: S 2998202135:2998202135(0) ack
  1207064129 win 5840 <mss 1460> (DF)
  05:09:52.927458 192.168.1.10.1211 > 10.40.100.152.80: R 1207064129:1207064129(0) win 0 (DF)
```

So the traffic is traversing the firewall and the webserver is responding, as expected based on the firewall rules.

Next I'll try the same thing only this time to port 25.

```
$ hping2 -V -p 25 -S 10.40.100.152
  using eth0, addr: 192.168.1.10, MTU: 1500
  HPING 10.40.100.152 (eth0 10.40.100.152): S set, 40 headers + 0 data bytes
```

This time the response is a more curt RST packet, but is it from the target, because it is not running Sendmail or the firewall? To try and test this I shall send another packet to port 25 only with a bad TCP checksum. Should I receive a reset, then I know that a firewall is filtering traffic, and also that it must be a fairly unsophisticated firewall.

```
$ hping2 -V -p 25 -S 10.40.100.152
  hping2 -V -p 25 -S 10.40.100.152 using eth0, addr: 192.168.1.10, MTU: 1500
  HPING 10.40.100.152 (eth0 10.40.100.152): S set, 40 headers + 0 data bytes
```

No reply at all this time, and thankfully the tcpdump running on the inside of the firewall registers no packets, so the packet was dropped at the firewall[12] which recognised the bad TCP checksum.

Defensive notes

This demonstration was meant more as a lead in to the next two perimeter probe techniques, so I shall not dwell on the analysis.

The majority of current firewalls should not be susceptible to this form of probe, as the demonstration proved. Even if the firewall was to take shortcuts and not examine the TCP checksum then silently dropping packets rather than sending resets would still evade this technique.

What it does teach us is that all anomalous packets should be treated as suspicious, and that assumptions have in the past and no doubt will be in the

[10] This is sniffing the inside of the firewall.

[11] I am not interested in any arps etc..

[12] This is confirmed by a reassuring 'DROP FORWARD tcp IN=eth0 OUT=eth1 SRC=192.168.1.10 DST=10.40.100.152 LEN=40 TOS=0x00 PREC=0x00 TTL=63 ID=16876 PROTO=TCP SPT=2162 DPT=25' message in our firewall l ogs

future, exploited by resourceful individuals.

2 Fire-walking

Fire-walking is a technique that can be used to gather information about a remote network protected by a firewall. In particular it is used to identify the filtering policy that the firewall implements, which is often different from the policy the firewall is intending to implement. In this regard it is a useful tool for both the security professional as well as the ardent hacker.

It is important to understand that this technique demonstrates to the user the 'open' ports on the gateway/firewall, and **not** the open ports on targeted hosts within the scrutinised network.

In order to understand the modus operandi of firewalking one needs analyse the IP TTL[13] field and it's practical exploitation by the traceroute program.

According to the Internet Protocol RFC791 the TTL field in an IP header ''indicates the maximum time the datagram is allowed to remain in the Internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in Internet header processing. ''.

What this means is that all conforming IP packets contain a timed self-destruct capability. Every time a packet passes through a compliant IP stack it's TTL field is decremented, and once it hits 0 the packet is annihilated. This is done before the packet is sent out again (presuming it is being routed on), and an ICMP time-exceeded in delivery packet is sent back to inform the source host of the demise of it's packet.

Traceroute takes advantage of this and starting with one, ramps up the TTL on consecutive packets[14]. These ill fated ICMP packet are intended to expire and generate from each router along the route to it's intended destination the ICMP 'timex'[15] packet. These 'timex' packets include the source address of the router sending them, so traceroute knows the packet got to router X after TTL hops. One after another it learns of the routers on the path to the intended destination. It knows that it has reached the end of the road when it receives an ICMP message indicating a UDP port unreachable.[16]

Basic premise is to send packets with a TTL of one more than the number of hops to the firewall that is being probed. If it passes the packet through then you will get an ICMP timeout message from the next hop.

Acquiring and compiling Firewalk

For this analysis I used the Firewalk utility written by Mike D. Schiffman, and the

[13] Time to live

[14] Actually most implementations of traceroute usuall y send packets in threes to be sure of the results . Remembering that packet delivery is not guaran teed by IP, that is the job of the transport protocols.

[15] ICMP 'Time Exceeded in Transit ' - Type 11, C ode 0

[16] This is the behavior of the Unix version of traceroute which sends a UDP packet to a port greater than 33000 (using V 1.4a12 on Linux traceroute ha s an initial port of 33435). Windows however uses an icmp echo-request packet and expects an echo reply when it reaches its intended target.

ubiquitous tcpdump as always for looking at the network traces.

Firewalk can be obtained from www.packetfactory.net, and is explained excellently in the paper "Firewalking A Traceroute-Like Analysis of IP Packet Responses to Determine Gateway Access Control Lists".

I used the same lab set up for this as for lab1, with the following firewall rules:

```
Chain INPUT (policy DROP)
target     prot opt source           destination
ACCEPT     tcp  --  10.40.100.121    10.40.100.159     tcp dpt:22
ACCEPT     udp  --  10.34.100.1      10.40.100.159     udp spt:53
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 8
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 0
DROP       udp  --  0.0.0.0/0        0.0.0.0/0         udp dpts:137:139
LOG        all  --  0.0.0.0/0        0.0.0.0/0         LOG level 4 prefix `DROP IN'

Chain FORWARD (policy DROP)
target     prot opt source           destination
ACCEPT     tcp  --  0.0.0.0/0        192.168.1.10      tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0         tcp spt:80 dpts:1025:65535
ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0         tcp spt:119 dpts:1025:65535
ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0         tcp spt:25 dpts:1025:65535
ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0         tcp spt:53 dpts:1025:65535
ACCEPT     udp  --  0.0.0.0/0        0.0.0.0/0         udp dpts:1025:65535
ACCEPT     udp  --  0.0.0.0/0        0.0.0.0/0         udp dpt:53
ACCEPT     udp  --  0.0.0.0/0        0.0.0.0/0         udp spt:53
LOG        tcp  --  0.0.0.0/0        0.0.0.0/0         LOG level 4 prefix `DROP FORWARD tcp'
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 8
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 0
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 11
DROP       udp  --  0.0.0.0/0        0.0.0.0/0         udp dpts:137:139
LOG        udp  --  0.0.0.0/0        0.0.0.0/0         LOG flags 0 level 4 prefix ` DROP
FORWARD udp packet'

Chain OUTPUT (policy ACCEPT)
target     prot opt source           destination
LOG        all  --  0.0.0.0/0        10.0.0.0/8        LOG level 4 prefix ` SPOOF ->Ext'
DROP       all  --  0.0.0.0/0        10.0.0.0/8
```

The major difference is that I am no longer sending resets but silently dropping the packets, as would more normally be the case. The rule set[17] is designed to mimic a simple border gateway device, such as a filtering router, many of which still do not have stateful inspection[18], as their primary function has historically been routing.

Firewalking in action

I first ran a standard nmap scan over the firewall. To reduce output and time, I limited it to scanning a very small subset of ports. I shall do this for all the future scans as well. A simple SYN scan such as this requires that the data actually gets to the target, unlike my firewalk scans. So I have to choose a host that I know is there. This would often limit me to the gateway itself, or an Internet facing server such as a webserver.[19] The best bet would seem the webserver,

[17] This rule set was designed in part to be as small as possible while allowing a reasonable demonstration of the firewalking tool. It is not intended to be a 'good' rule set.

[18] I have not therefore used stateful inspection here, inst ead relied on a philosophy of allowing in bound traffic on from well known ports to ephemeral ports on the assumption it is return traffic. I stress again this is a lab rule set only.

[19] I would no doubt be able to obtain the address for th is by a DNS lookup of **www.target-domiain**.

and in my lab that is 10.40.100.152.

```
# nmap -sS -p 22,25,53,80 10.40.100.152
  Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
  Interesting ports on  (10.40.100.152):
  Port       State       Service
  22/tcp     filtered    ssh
  25/tcp     filtered    smtp
  53/tcp     filtered    domain
  80/tcp     filtered    http
  1025/tcp   filtered    listen
  Nmap run completed -- 1 IP address (1 host up) scanned in 57 seconds
```

Based on this the hacker might assume that the security policy was to filter ingress traffic to these ports, and move on to a more inviting target. What would a firewalk show them?

I next ran firewalk to scan for the ports 22, 25, 80, 53 and 1025 using TCP port 80 and then UDP port 53 as the source port.[20] I have to provide firewalk with a destination host as well as the gateway address I intend to scan. Importantly though the traffic does not have to ever reach this host, in fact it could be off, or even non-existent.[21] I chose a random IP in the 10.34.100.0 subnet.[22]

```
# firewalk -S 22,25,53,80,1025 -p tcp -s 80 192.168.1.100 10.34.12.21
  1 (TTL  1): port  22: Firewalk 5.0 [gateway ACL scanner]
  Firewalk state initialization completed successfully.
  TCP-based scan.
  Ramping phase source port: 80, destination port: 33434
  Hotfoot through 192.168.1.100 using 10.34.12.21 as a metric.
  Ramping Phase:
  expired [192.168.1.100]
  Binding host reached.
  Scan bound at 2 hops.
  Scanning Phase:
  *no response*
  port  25: *no response*
  port  53: *no response*
  port  80: *no response*
  port 1025: open (expired) [10.40.100.170]
```

This first scan using TCP and a source port of 80, shows us that the firewall is allowing traffic in from port 80 to 10.34.12.21 on port 1025. From this we could guess that there is a rule allowing return web traffic in to any host on an ephemeral port. We would want to run the probe again using a different ephemeral port and target to be sure, but it would seem unlikely that we randomly chose a host that is specifically allowed traffic through. We also know that the 10.34.12.21 machine is in a different network to our gateway host, with a router 10.40.100.170 in between.

Lets see what our next scan gave us.

```
# firewalk -S 22,25,53,80,1025 192.168.1.100 10.34.12.21
   1 (TTL  1): port  22: port  25: port  53: port  80: port 1025: Firewalk 5.0 [gateway ACL
  scanner]
```

[20] Again this is to save some time and limit the amount of output. Under normal circumstances I might scan all ports. The use of 53 and 80 as a source port is to see if return traffic from DNS and web servers is allowed in.

[21] It does need to be routable, and preferably more than one hop from the gateway.

[22] The results would have been similar had I chosen any IP in any subnet of the 10.0.0.0/8 internal network other than 10.40.100.0/24. This subnet is on the other side of the filtering router so my packets would never have expired, this can be accommodated for by setting an option in firewalk. It then acts more like a standard scanner, looking fore resets etc..

```
Firewalk state initialization completed successfully.
UDP-based scan.
Ramping phase source port: 53, destination port: 33434
Hotfoot through 192.168.1.100 using 10.34.12.21 as a metric.
Ramping Phase:
expired [192.168.1.100]
Binding host reached.
Scan bound at 2 hops.
Scanning Phase:
open (expired) [10.40.100.170]
open (expired) [10.40.100.170]
open (expired) [10.40.100.170]
open (expired) [10.40.100.170]
open (expired) [10.40.100.170]
```

Defensive notes

That is all well and good was this a study on network mapping, but I am more interested in what we can learn as IDS analysts. Lets look at a trace and see if there is anything we can use.

```
#tcpdump -nvxXi eth1
  07:04:57.036605 192.168.1.10.80 > 10.34.12.21.33434: S [tcp sum ok] 1274816059:1274816059(0)
  win 1024 [ttl 1] (id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0106 b125 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 829a 4bfc 263b 0000 0000              ."...P..K.&;....
  0x0020 5002 0400 ded7 0000 0331 3030 0131                   P........100.1
  07:04:57.036720 192.168.1.100 > 192.168.1.10: icmp: time exceeded in-transit [tos 0xc0]  (ttl
  255, id 11512, len 68)
  0x0000 45c0 0044 2cf8 0000 ff01 0a42 c0a8 0164              E..D,......B...d
  0x0010 c0a8 010a 0b00 cd03 0000 0000 4500 0028              ............E..(
  0x0020 30c2 0000 0106 b125 c0a8 010a 0a22 0c15              0......%.....".."
  0x0030 0050 829a 4bfc 263b 0000 0000 5002 0400              .P..K.&;....P...
  0x0040 ded7 0000                                            ....
  07:04:57.036851 192.168.1.10.80 > 10.34.12.21.22: S [tcp sum ok] 1274816059:1274816059(0) win
  1024 (ttl 2, id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0206 b025 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 0016 4bfc 263b 0000 0000              ."...P..K.&;....
  0x0020 5002 0400 615c 0000 0c0d 0e0f 1011                   P...a\........
  07:04:59.035149 192.168.1.10.80 > 10.34.12.21.25: S [tcp sum ok] 1274816059:1274816059(0) win
  1024 (ttl 2, id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0206 b025 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 0019 4bfc 263b 0000 0000              ."...P..K.&;....
  0x0020 5002 0400 6159 0000 0c0d 0e0f 1011                   P...aY........
  07:05:01.035078 192.168.1.10.80 > 10.34.12.21.53: S [tcp sum ok] 1274816059:1274816059(0) win
  1024 (ttl 2, id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0206 b025 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 0035 4bfc 263b 0000 0000              ."...P.5K.&;....
  0x0020 5002 0400 613d 0000 0c0d 0e0f 1011                   P...a=........
  07:05:03.035016 192.168.1.10.80 > 10.34.12.21.80: S [tcp sum ok] 1274816059:1274816059(0) win
  1024 (ttl 2, id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0206 b025 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 0050 4bfc 263b 0000 0000              ."...P.PK.&;....
  0x0020 5002 0400 6122 0000 0331 3030 0131                   P...a"...100.1
  07:05:05.034952 192.168.1.10.80 > 10.34.12.21.1025: S [tcp sum ok] 1274816059:1274816059(0)
  win 1024 (ttl 2, id 12482, len 40)
  0x0000 4500 0028 30c2 0000 0206 b025 c0a8 010a              E..(0......%....
  0x0010 0a22 0c15 0050 0401 4bfc 263b 0000 0000              ."...P..K.&;....
  0x0020 5002 0400 5d71 0000 0c0d 0e0f 1011                   P...]q........
  07:05:05.036967 10.40.100.170 > 192.168.1.10: icmp: time exceeded in-transit (ttl 254, id
  33971, len 56)
  0x0000 4500 0038 84b3 0000 fe01 078d 0a28 64aa              E..8.........(d.
  0x0010 c0a8 010a 0b00 7e77 0000 0000 4500 0028              ......~w....E..(
  0x0020 30c2 0000 0106 b125 c0a8 010a 0a22 0c15              0......%.....".."
  0x0030 0050 0401 4bfc 263b                                  .P..K.&;
```

There are a number of things that stand out when looking at this trace. Firstly the repeating patterns within packets, such as the constant IPID, and the trailing **0c0d 0e0f 1011.** The latter could well be frame padding as it is after the reported length of the packet. The former is definite indication of packet craft. However a signature based on this could easily be thwarted by a minor modification to the code. It would be better if we could trigger on a pattern that is an intrinsic part of the probes behavior rather than a coding aberration.

It is important to realise that this is the trace of two, separate phases of the probe. The initial phase in this case spans the first two packets and is the 'ramping phase. Using the default settings the tool behaves almost identically to the Unix traceroute tool, sending out UDP packet to a high port with a low TTL looking for a time out. In the trace above I was using TCP port 80 which would be easier to spot.

The second phase is the actual scan, which utilises the TTL calculated in the ramping phase to ensure the probe packets expire one hop after the gateway.[23] Again we can see some patterns in the packet such as the static IPID and SEQ number, which would be used to generate a signature. As well as this there is the TTL of two, as the packet enters the gateway. Under normal circumstances the TTL is used to identify routing loops and should not be this low, unless it is part of a traceroute. Can we use this to our advantage? A traceroute would not be sending a SYN packet to port 22! Presuming your network has an internal max distance of x, we could look for incoming packets of less that x+1. So assuming a value of 5 for x:

Proposed Snort Rule:

```
alert tcp $EXTERNAL_NET ANY -> $HOME_NET :33000[24] (msg:"Suspicious
TTL, possible Firewalk attempt";ttl:<6; classtype:attempted -recon;)
```

This rule should trigger on any packet entering our network to a port less than 33,000 with TTL of less than 6. Lets see if it is going to detect our firewalk.[25]

Additionally to detecting firewalk attempts, you should be looking to stop it. Blocking egress time exceeded packets is one way[26], filtering traffic on destination IP as well as port is another. That way the hacker would have to guess the IP of the service, which makes the whole probe a little futile. Defense in depth will also make the hacker's job harder, as packets may be dropped by a filtering router, which tells the hacker nothing about the firewall's rules. Using a private RFC1918 address block for your internal network would also help if everything was NATd behind one Internet address, as the firewalk requires an internal address which is routable from the internet for it to work properly.

[23] This can be modified to be greater than one by command line options.

[24] Unix traceroute usually sends packets to ports greater than 33,000, as there is little chance of a service listening on such a high port.

[25] I tested this in the lab set up and it did successfully detect the firewalk, while not generating false positives on traceroutes etc..

[26] This would also stop traceroute from being able to map route to your internal servers, but is that such a bad thing?

3 Identifying Trust Relationships

Trust relationships can be exploited to great effect by a hacker who is targeting your network, as demonstrated by Kevin Mitnick's attack against Tsutomu Shimomura's system. Mitnick *"detected a trust relationship between two computers and exploited that relationship"*[27]

A Trust relationship is one whereby privileges or access to one host is granted to another host simply by virtue of who it is. This was what Mitnick was able to detect between one of Shimomura's X-terminal workstations and one of his servers. By impersonating the X-terminal he was able to execute a number of remote commands [28]without any need to authenticate himself. The trick for the attacker is to identify these trust relationships in order to later exploit them.

IPID Scans

There are a number of techniques available to identify trust relationships, which vary from simple social engineering, to more technical network probes and passive network analysis. I will concentrate on one method, which uses an active network probe.

No talk on network reconnaissance could be complete without mentioning Nmap[29] developed by Fyodor. In his own words *"Nmap ("Network Mapper") is an open source utility for network exploration or security auditing."* It is an extremely rich tool which is often simply used to determine what ports are open on a particular host, but it's functionality is much greater than this. Amongst it's myriad of features is a implementation of Antirez' so called idle scan[30]. The idle scan is an extremely stealthy port scan where the scanner sends his crafted packets to the target but spoofs the source IP to be of a 3rd party. I shall refer to this 3rd party host as our 'zombie.' This allows his activity to go on without the victim being able to correctly identify the true source. An important factor to note in this description is that the port scan is therefore done from the perspective of the zombie. The offshoot of all this is that by careful choice of the zombie our hacker can start to map out possible trust relationships.

eg. Choose the zombie to be the address of the targets external webserver etc...

Idle Scan in action

In this Demonstration I will use the following network which contains a trust relationship between 'Server A' and 'Workstation B'. In this case unlike the Mitnick attack I have chosen the SSH protocol. This is a much more secure administration tool as it provides both strong authentication of both client and

[27] Northcutt, Stephen & Novak, Judy - Network Intrusion Detection, an Analyst's Handbook - Second Edition

[28] The 'r-commands' are a number of commands such as rsh - remote shell, rcp - remote copy, rlogin - remote login etc.. which are listed as number six in SANS Unix top vulnerabilities list (May 5 2003). The are unencrypted so the content can be sniffed in the clear and suffer from poor host authentication.

[29] www.insecure.org

[30] First identified in a posting made by Salvatore Sanfilippo of Intesis SECURITY LAB in Dec 17 1998 outlining the technicalities of such a probe.

server as well as an encrypted data channel. In my example however I am looking at perimeter mapping trust relationships. In this scenario the internal Server is protected by a firewall, however SSH traffic is allowed in to it from the administrators home workstation to facilitate after hours troubleshooting. Should an SSH vulnerability be found such as CERT® Advisory CA-2002-18[31] then the internal server is vulnerable should an attacker discover this trust relationship. Often it is more vulnerable services such as FTP access to the companies internal webserver etc...

The networks used in this lab are all non internet routable RFC 1918 addresses for test purposes. The two external workstations are on the same segment in this example which would make the exploitation of a trust relationship easier. However I am solely focusing on the discovery of trust relationships as a part of perimeter mapping. The technique I am investigating does not rely on the two machines being on the same network segment.

In this setup all hosts were running Debian Linux (2.4 kernel). The IDS sensor had it's interfaces running in promiscuous mode and was not configure with an IP address. The router is filtering using iptables with the following configuration:

```
Chain INPUT (policy DROP)
target     prot opt source           destination
ACCEPT     tcp  --  10.40.100.121    10.40.100.159     tcp dpt:22
ACCEPT     udp  --  10.34.100.1      10.40.100.159     udp spt:53
ACCEPT     icmp --  0.0.0.0/0        10.40.100.159
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 8
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0         icmp type 0
DROP       udp  --  0.0.0.0/0        0.0.0.0/0         udp dpts:137:139
LOG        all  --  0.0.0.0/0        0.0.0.0/0         LOG level 4 prefix `DROP IN'

Chain FORWARD (policy DROP)
target     prot opt source           destination
ACCEPT     tcp  --  0.0.0.0/0        10.40.100.152      tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0          state ESTABLISHED
ACCEPT     udp  --  0.0.0.0/0        0.0.0.0/0          state ESTABLISHED
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0          icmp type 8
ACCEPT     icmp --  0.0.0.0/0        0.0.0.0/0          icmp type 0
ACCEPT     tcp  --  192.168.1.150    10.40.100.152      tcp dpt:22
ACCEPT     tcp  --  192.168.1.150    10.40.100.152      tcp dpt:20
ACCEPT     tcp  --  192.168.1.150    10.40.100.152      tcp dpt:21
ACCEPT     tcp  --  0.0.0.0/0        10.34.100.2        tcp dpt:25
ACCEPT     udp  --  192.168.5.1      10.34.100.1        udp dpt:53
ACCEPT     tcp  --  192.168.5.1      10.34.100.1        tcp dpt:53
LOG        tcp  --  0.0.0.0/0        0.0.0.0/0          LOG level 4 prefix `DROP --> tcp'
LOG        udp  --  0.0.0.0/0        0.0.0.0/0          LOG level 4 prefix `DROP --> udp'
LOG        icmp --  0.0.0.0/0        0.0.0.0/0          LOG level 4 prefix `DROP --> icmp'

Chain OUTPUT (policy ACCEPT)
target     prot opt source           destination
LOG        all  --  0.0.0.0/0        10.0.0.0/8         LOG level 4 prefix ` SPOOF ->Ext '
DROP       all  --  0.0.0.0/0        10.0.0.0/8
```

As can be seen from the above rule set the firewall/router is allowing traffic in to port 22, and ports 21 and 20 on the webserver from the external management host. These are of course the 'Well Known Ports' for SSH and FTP as allocated by IANA[32]. No other traffic is allowed in except some limited ICMP traffic, and

[31] CERT® Advisory CA-2002-18 OpenSSH Vulnerabilities in Challenge Response Handling - December 6, 2002.

[32] Internet Assigned Numbers Authority. - http://www.iana.org/ §

traffic that is part of an ESTABLISHED outgoing session.

A standard SYN scan of this perimeter would show all incoming TCP traffic being blocked except HTTP destined for the webserver, as demonstrated in the following scan.

Standard Nmap scan of internal network.

```
#nmap -sS -F 10.40.100.152 [33]
  Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
  Interesting ports on  (10.40.100.152):
  (The 1099 ports scanned but not shown below are in state: filtered)
  Port       State       Service
  80/tcp     open        http

  Nmap run completed -- 1 IP address (1 host up) scanned in 910 seconds
```

As expected the Syn scan came up empty apart from the allowed web traffic. What about an idle scan using the management host as a zombie?

Idle Nmap scan of internal network.[34]

```
#nmap -F -sI 192.168.1.150  10.40.100.152
  Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
  Idlescan using zombie 192.168.1.150 (192.168.1.150:80); Class: Incremental
  Interesting ports on  (10.40.100.152):
  (The 1097 ports scanned but not shown below are in state: closed)
  Port       State       Service
  21/tcp     open        ftp
  22/tcp     open        ssh
  80/tcp     open        http

  Nmap run completed -- 1 IP address (1 host up) scanned in 920 seconds
```

So from the perspective of the chosen 'zombie' the firewall is not as daunting. The question is how can we as analysts protect ourselves from such scans? The problem an IDS analyst faces is that this traffic is going to look like expected management traffic, so if the scanner is patient enough, it could be very hard for the analyst to spot. Even if a pattern is observed, tracking the true source of the traffic will be extremely difficult.

Analysis of traces

First there is a small flurry of SYNs from the scanning host and subsequent RSTs from the proposed zombie. This is to determine if we can reliably predict the IPID.

```
# tcpdump -nvi eth1
  05:52:08.427456 192.168.1.10.45473 > 192.168.1.150.1025: S [tcp sum ok]
  1210372252:1210372252(0) ack 0 win 23826 (ttl 58, id 28336, len 40)
  05:52:08.427573 192.168.1.150.1025 > 192.168.1.10.45473: R [tcp sum ok] 0:0(0) win 0 (ttl
  128, id 55571, len 40)
  05:52:08.466501 192.168.1.10.45474 > 192.168.1.150.1025: S [tcp sum ok]
  1210372253:1210372253(0) ack 0 win 23826 (ttl 58, id 4912, len 40)
  05:52:08.466588 192.168.1.150.1025 > 192.168.1.10.45474: R [tcp sum ok] 0:0(0) win 0 (ttl
  128, id 55572, len 40)
```

[33] -sS indicates a Syn scan, -F sets nmap to inly scan ports listed in the /etc/services file, and is much faster than scanning all 65,535 possible ports.

[34] I used nmap by Fyodor as it is a highly stable network scanner, and it also i mplements IPID scans. In the demonstrations I took advantage of the fact that I knew the internal network to speed up the sca ns and increase the readability of the output. It does not effect the results.

```
05:52:08.506499 192.168.1.10.45475 > 192.168.1.150.1025: S [tcp sum ok]
1210372254:1210372254(0) ack 0 win 23826 (ttl 58, id 35997, len 40)
05:52:08.506585 192.168.1.150.1025 > 192.168.1.10.45475: R [tcp sum ok] 0:0(0) win 0 (ttl
128, id 55573, len 40)
05:52:08.546499 192.168.1.10.45476 > 192.168.1.150.1025: S [tcp sum ok]
1210372255:1210372255(0) ack 0 win 23826 (ttl 58, id 51387, len 40)
05:52:08.546585 192.168.1.150.1025 > 192.168.1.10.45476: R [tcp sum ok] 0:0(0) win 0 (ttl
128, id 55574, len 40)
05:52:08.586495 192.168.1.10.45477 > 192.168.1.150.1025: S [tcp sum ok]
1210372256:1210372256(0) ack 0 win 23826 (ttl 58, id 32491, len 40)
05:52:08.586582 192.168.1.150.1025 > 192.168.1.10.45477: R [tcp sum ok] 0:0(0) win 0 (ttl
128, id 55575, len 40)
05:52:08.626494 192.168.1.10.45478 > 192.168.1.150.1025: S [tcp sum ok]
1210372257:1210372257(0) ack 0 win 23826 (ttl 58, id 39914, len 40)
05:52:08.626579 192.168.1.150.1025 > 192.168.1.10.45478: R [tcp sum ok] 0:0(0) win 0 (ttl
128, id 55576, len 40)
```

After this the probe in earnest begins. The probes to our network are interspersed with SYN RST pairs from the scanning host to the zombie, as illustrated below in the following tcpdump extract:

```
05:52:09.446473 192.168.1.10.45602 > 192.168.1.150.1025: S 4007888134:4007888134(0) ack
899391217 win 3072
05:52:09.446559 192.168.1.150.1025 > 192.168.1.10.45602: R 899391217:899391217(0) win 0
05:52:09.446660 192.168.1.150.1025 > 10.40.100.152.22: S 897127136:897127136(0) win 3072
05:52:09.446725 192.168.1.150.1025 > 10.40.100.152.53: S 897127136:897127136(0) win 3072
05:52:09.463312 10.40.100.152.22 > 192.168.1.150.1025: S 60877630:60877630(0) ack 897127137
win 5840 <mss 1460> (DF)
05:52:09.463398 192.168.1.150.1025 > 10.40.100.152.22: R 897127137:897127137(0) win 0
05:52:09.506466 192.168.1.10.45717 > 192.168.1.150.1025: S 4007888634:4007888634(0) ack
899391217 win 3072
05:52:09.506552 192.168.1.150.1025 > 192.168.1.10.45717: R 899391217:899391217(0) win 0
```

The scanning system is looking to see if the IPID has changed, indicating a reply from the target to our zombie.

Looking at the extract for the entire scan this pattern is fairly regular and easy to spot. I was able to collect this data due to the fact that both external hosts were on the same segment, so my IDS sensor saw all the traffic. This would not normally be the case so what is the solution? This brings to the fore two aspects of IDS forensics I feel are often neglected.

1. Your Firewall/application/Operating system logs are all useful Intrusion detection tools do not forget them.
2. Correlation of data from multiple sources can be invaluable for spotting stealthy attacks and probes.

How does all this help our fight against IPID scans? Firstly if you must have a trust relation between hosts, it is imperative that you take active steps to protect both machines! It is no use firewalling you webserver if all an attacker needs is to compromise an external host to gain access via a trust relationship. Log all system, ids and firewall logs, to a central place to allow correlation of data[35]. The pattern above is hard enough to spot in one log, let alone split into many separate logs on different servers that are not time synced![36]

Using some form of stateful log analysis tool such as 'logsurfer[37]' will aid in the

[35] Try not to allow this to be exploited for a possibl e DOS of your analysis capabilities however.

[36] Implementing a standard time is essential for a useful log correlation implementation, signals form GPS satellites propagated via NTP is one solution.

[37] 'logsurfer' is a log watching tool similar to swatch. It i s stateful in it can anal yse a log line in the context of preceding lines etc.. and can add dynamic rules.

ability to spot patterns such as the one above, provided the logs are collected appropriately.

References

1.CERT® Advisory CA-2002-18 OpenSSH Vulnerabilities in Challenge Response Handling - December 6, 2002. - http://www.cert.org/advisories/CA-2002-18.html (May 05 2003)

2.'Ed3f' - Firewall spotting and networks analysis with a broken CRC', Phrack Volume60 URL:http://www.phrack.org/show.php?p=60&a=12 (5 April 2003)

3.Fyodor, Insecure.org - Idle Scanning and related IPID games URL: http://www.insecure.org May 05 2003)

4.Goldsmith , David and Schiffman ,Michael - Firewalking A Traceroute-Like Analysis of IP Packet Responses to Determine Gateway Access Control Lists, URL:http://secinf.net/info/ids/idspaper/idspaper.html (May 05 2003)

5.IANA well known ports listing http://www.iana.org/assignments/port-numbers (May 05 2003)

6.Ley, Wolfgang and Ellerman, Uwe - Logsurfer URL:http://www.cert.dfn.de/eng/logsurf/ (July 12 2003)

7.Northcutt, Stephen & Novak, Judy - 'Network Intrusion Detection, an Analyst's Handbook - Second Edition' – NewRiders 2001

8.Postal, John - RFC 793 Transmission Control Protocol URL:http://www.faqs.org/rfcs/rfc793.html (May 05 2003)

9.Ptacek , Thomas H. and Newsham , Timothy, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Secure Networks, January 1998 URL:http://www.insecure.org/stf/secnet_ids/secnet_ids.html (20 March 2003)

10.RFC791 - Internet protocol, DARPA internet program protocol specification URL:http://www.faqs.org/rfcs/rfc791.html (April 20 2003)

11.Sanfilippo, Salvatore - hping.org URL:http://www.hping.org/ (July 8 2003)

12.SANS Top Vulnerabilities –URL: http://www.sans.org/top20/#U6 (May 05 2003)

13.Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994.

Assignment 2 - Net Detects

Posted: https://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00060.html

The following three net detects highlight a number of things about intrusion detection.

Firstly false positives are everywhere, but still need to be analysed. In detect one I look at a packet which I initially thought to be crafted, but with greater analysis proves a better candidate for a false positive. It is taken from incidents.org.

Secondly correlation of data can be invaluable in determining the true nature of an IDS detect. In detect two I look at a fairly simple trace of some scanning, and

use additional data sources such as firewall logs. It is taken from my home network.

Analysis one.

1. Source of Trace.

This detect was extracted from the following raw log downloaded off the GIAC assignment site as per the instructions in the current assignment 3.3. 'Raw/2002.5.10.log'. From the readme file at incidents.org, '/logs/Raw/README' I noted that the checksum errors found in the tcpdump and snort outputs are a result of the sanitisation process these raw logs were put through prior to being posted.

The first time stamp in the data file is '12:18:48' and the last is ' 11:52:36' this indicates a time period of roughly 24 hours for the data collected on Friday the 10 May 2002 as indicated by the file name.

I used the following snort command to generate an alert file from the binary file.

```
# snort -c snort.conf -l ./ -r 2002.5.10
```

The alert entry that I decided to investigate was:

```
 [**] [1:523:3] BAD TRAFFIC ip reserved bit set [**]
[Classification: Misc activity] [Priority: 3] 06/11 -01:41:19.544488
218.2.129.171 -> 46.5.188.185 TCP TTL:231 TOS:0x0 ID:0 IpLen:20
DgmLen:40 RB Frag Offset: 0x11F1 Frag Size: 0xFFFFEE23
```

In order to find the packets that had generated this alert I used tcpdump to find any packets to or from 218.2.129.171.

```
# tcpdump -vvr 2002.5.10.log host 218.2.129.171
01:41:19.544488 218.2.129.171 > 46.5.188.185: (frag 0:20@36744)
(ttl 231, len 40, bad cksum 178!)
06:51:14.694488 218.2.129.171 > 46.5.142.232: (frag 0:20@32+) (ttl
231, len 40, bad cksum a136!)
```

Network Layout

Since the trace was taken from the incidents.org web site as a single snort binary log there is no way of knowing for sure any details of the originating network's topology. One can however examine the data and make some observations about a probable topology.

First I examined the file for a list of unique source and destination MAC addresses[38].

To do this I used a combination of awk, tcpdump, sort and uniq all common Unix commands to isolate the unique source and destination MAC addresses.

Tcpdump when given the -e flag will print the link-level header on each dump line. The format of the output means that the second and third tokens are the source and destination mac addresses respectively. eg.

[38] **M**edia **A**ccess **C**ontrol address, a hardware address that uniquely identifies each node of a network. - http://www.webopedia.com

```
13:02:17.164488  0:3:e3:d9:26:c0  0:0:c:4:b2:33  ip 1514:
203.177.0.39.www > 46.5.180.250.62119: . [bad tcp cksum f9f9!]
1814404202:1814405662(1460) ack 247126970 win 7765 (DF) (ttl 110,
id 10401, len 1500, bad cksum 35a9!)
```

I was therefore able to use the following command to extract the unique source mac addresses for all the packets in the log file:

```
# tcpdump -er 2002.5.10.log | awk '{print $2}' | sort  -n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

The following command was used to get a list of unique destination mac addresses:

```
# tcpdump -er 2002.5.10.log | awk '{print $3}' | sort  -n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

This shows that there are only two unique MAC addresses in the log file which suggests that the sniffer or ids probe was probably placed either between a perimeter router and a firewall, or the perimeter router/firewall and the ISPs perimeter router/firewall.

Next I tried to identify the MAC address using the searchable MAC address/Vendor database at http://www.coffer.com/mac_find/. By using the first three octets of the MAC addresses I was able to determine that both belonged to Cisco hardware.

Further investigation of the traffic showed that there were 146 unique destination addresses all in the 46.5.0.0/16 range.

```
#tcpdump -vn -r 2002.5.10.log | awk '{print $4}' | sed  s/ \\.[0-9]*:// |
sort -n | uniq | grep ^46.5| wc  -l
  146
```

This includes the network address 46.5.0.0, so a more accurate figure is 145. This compares to a total of 234 unique destination addresses. This is augmented by the fact that a whois search[39] for this address revealed that it is a reserved network.

Trying whois -h whois.arin.net 46.5.0.0 OrgName: Internet Assigned Numbers Authority
OrgID: IANA Address: 4676 Admiralty Way, Suite 330
City: Marina del Rey
StateProv: CA
PostalCode: 90292-6695
Country: US
NetRange: 46.0.0.0 -46.255.255.255
CIDR: 46.0.0.0/8
NetName: RESERVED-46
NetHandle: NET-46-0-0-0-0
Parent:
NetType: IANA Reserved
Comment:
RegDate:
Updated: 2002-08-23

 OrgTechHandle: IANA-ARIN
OrgTechName: Internet Corporation for Assigned Names and Number
OrgTechPhone: +1-310-823-9358
OrgTechEmail: res-ip@iana.org
ARIN WHOIS database, last updated 2003-06-05 21:05

[39] I used www.samspade.org to perform this lookup.

```
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

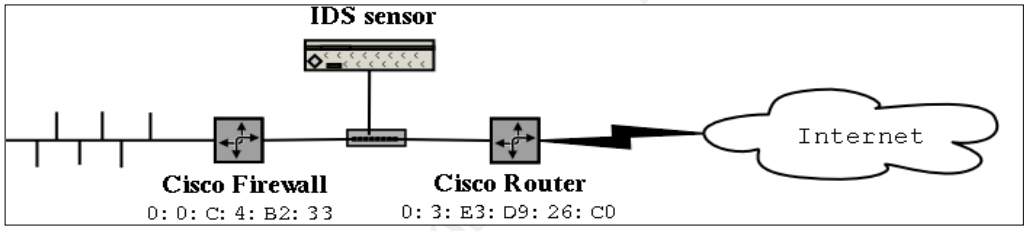
We know from the accompanying README file[40] that the protected network has had it's IP addresses obfuscated to a different network. This seems to confirm our suspicions that this is indeed the protected network.

All packets destined to this address range have a destination MAC address of, 0:0:C:4:B2:33.

```
$  tcpdump -ner 2002.5.10.log 'src net 46.5.0.0/16'  | wc  -l
   3860
$  tcpdump -ner 2002.5.10.log 'ether src 0:0:c:4:b2:33 and src net
46.5.0.0/16'  | wc  -l
   3860
$  tcpdump -ner 2002.5.10.log 'ether dst 0:0:c:4:b2:33 and dst net
46.5.0.0/16'  | wc  -l
    448
$  tcpdump -ner 2002.5.10.log 'dst net 46.5.0.0/16'  | wc  -l
    448
```

It seems logical therefore to assume that this is the address range of the target network[41] situated behind the proposed Cisco router/firewall (0:0:C:4:B2:33).

Based on this information a probable network layout may well be as follows.



Analysis of the traffic using p0f[42] might reveal the type of hosts in this network.

It is important to note that p0f only looks at SYN packets for the tell tale IP and TCP details which act as a signature for a particular operating system. It could therefore shed no light upon the source of our suspicious packet which is a fragment. I used the following command to see if I could find out any more interesting data about the target network using p0f.

```
$ p0f -s 2002.5.10.log
p0f: passive os fingerprinting utility, version 1.8.3
(C) Michal Zalewski <lcamtuf@gis.net>, William Stearns <wstearns@pobox.com>
p0f: file: '/etc/p0f.fp', 207 fprints, iface: 'lo', rule: 'all'.
64.228.63.154 [15 hops]: Linux 2.4.2  - 2.4.14 (1)
```

Unfortunately this was the only host p0f was able to identify from the log file, and it is not within the suspected interior network of 46.5.0.0/16.

[40] See Apendix A

[41] This is not the true address range though as this has been modified to protect the identity of the target network. All addresses in this file have been modified in a consistent manner according to the accompanying text file.

[42] p0f is a remote passive system fingerprinting tool written by Michal Zalewski < lcamtuf@coredump.cx> .

2. Detect was generated by:

Snort Intrusion detection system.

version 1.9

Rule:

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC ip
reserved bit set"; fragbits:R; sid:523;  classtype:misc -activity;
rev:3;)
```

3. Probability the source address was spoofed:

There is no evidence of source address spoofing that can be gathered from this packet. So we are forced to rely upon motive. This is discussed in great detail in section five, where I will conclude that this packet is probably non-malicious. Given this assumption the probability that the packet has it's source address spoofed is negligible. The source address is routable, and an allocated address used by Chinanet-js, a large Chinese telecom subsidiary, according to a 'whois' search.[43] The only evidence to go against this could be the lack of other packets from this IP address. This can be discounted however, as the logs in question only contain packets that triggered the IDS, so the other 'good' packets may well have been present but obviously did not trigger the sensors.

4. Description of attack:

Initial log alert entry in 2002.5.10.log

```
[**] [1:523:3] BAD TRAFFIC ip reserved bit set [**]
[Classification: Misc activity] [Priority: 3] 06/11 -01:41:19.544488
218.2.129.171 -> 46.5.188.185 TCP TTL:231 TOS:0x0 ID:0 IpLen:20
DgmLen:40 RB Frag Offset: 0x11F1 Frag Size: 0xFFFFEE23
```

```
# tcpdump -vvr 2002.5.10.log host 2 18.2.129.171
01:41:19.544488 218.2.129.171 > 46.5.188.185: (frag 0:20@36744)
(ttl 231, len 40, bad cksum 178!)
06:51:14.694488 218.2.129.171 > 46.5.142.232: (frag 0:20@32+) (ttl
231, len 40, bad cksum a136!)
```

packet dump

```
# tcpdump -vvxXr 2002.5.10.log host  218.2.129.171
01:41:19.544488 218.2.129.171 > 46.5.188.185: (frag 0:20@36744)
(ttl 231, len 40, bad cksum 178!)
0x0000 4500 0028 0000 91f1 e706 0178 da02 81ab E  ..(.......x....
0x0010 2e05 bcb9 8329 0050 02fa f904 02fa f904    .....).P........
0x0020 5004 0000 f402 0000 0000 0000 0000      P ..............

06:51:14.694488 218.2.129.171 > 46.5.142.232: (frag 0:20@32+) (ttl
231, len 40, bad cksum a136!)
0x0000 4500 0028 0000 2004 e706 a136 da02 81ab E  ..(.......6....
0x0010 2e05 8ee8 81db 0050 0416 b6e8 0416 b 6e8   .......P........
0x0020 5004 0000 a522 0000 0000 0000 0000      P  ....".........
```

Second packet we notice triggered on

[43] An Internet utility that returns information about a dom ain name or IP address., the results of which can be seen in the Correlation section.

```
[**] [1:522:1] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
06/11-06:51:14.694488 218.2.129.171 -> 46.5.142.232
TCP TTL:231 TOS:0x0 ID:0 IpLen:20 DgmLen:40 MF
Frag Offset: 0x0004 Frag Size: 0x0010
```

Looking at IP[6] on each of these packet we can see that the second packet is OK as far as the reserved bit is concerned. Masking this byte with 0xE0 gives us the 3 bits we are interested in, the IP flags. The two low order bits are the MF and DF flags (More Fragments and Don't Fragment) respectively. The high order bit however is the infamous reserved bit (according to RFC-791 "Bit 0: reserved, must be zero").

The 6th and 7th bytes are 0x91f1, but the first nibble also contains the reserved bit. Closer inspection of this nibble reveals the following bit pattern:

```
1001
```

So the first packet has a fragment offset of 0x11f1 or 4593 (8 byte words), or 36744 bytes. It contains however, only 20 bytes of data, or does it? The total size of the packet is 0x28, or 40 bytes. The IP header is 5*(4 byte words) or 20 bytes, so this leaves 20 bytes for the payload (in this case a fragment). I presume this includes some padding, as the last four bytes of the packet are all zeros. Why would such a large packet get fragmented so small, unless it was the last packet? The MF bit is not set which would indicate that this is the last fragment (the so called runt). The Fragment ID of 0 is somewhat suspicious though.

There are a number of circumstances under which the Linux IP stack (up to kernel 2.4.5) would always create a packet with an IP ID of zero[44].

1. ICMP: Kernel 2.4.0-2.4.4 will use the value of zero (0) for the IP ID field value whenever sending an ICMP query messages or producing ICMP replies. This behavior was changed with Kernel 2.4.5 and above, and now only when generating ICMP query messages the IP ID field value will be set to zero.
2. Whenever sending or answering for a UDP datagram the IP ID will be zero when the DF bit will be[sic] set.
3. TCP: In several circumstances, like a SYN-ACK answer for a SYN, the IP ID will be zero when the DF bit will be[sic] set.

However as we have indicated this is not likely to be an ICMP packet, and obviously the DF bit is not set as this is a fragmented packet! So the IP ID remains somewhat suspicious.

The Reserved bit on packet one is set which is what triggered this alert, yet according to the RFC for IP (RFC791) this bit should always be zero. I shall examine possible reasons for this is my discussion of the attack mechanism.

[44] The following bullet points are excerpts from 'A crash course with Linux Kernel 2.4.x, IP ID values; RFC 791' - Ofir Arkin (13 Apr 2002)

5.Attack mechanism:

Possibility one: Abnormal[45] packets as a form of OS identification.

This is one possible explanation for this packet. Common tools such as nmap will use abnormal packets as stimuli to attempt to identify a remote OS, based on the lack of conformity in the stack implementations of various OSs. An example of such an abnormal packet is the 'reserved bit set' packet.

In a post to Nmap Hacker, Ofir Arkin put forward the use of such a technique.[46] In this post he demonstrated the possibilities by differentiating Sun Solaris and OpenBSD replies from HPUX 11.0 responses. This analysis was made based simply on the reply to an ICMP echo-request packet with the reserved bit set. Interesting though this is, it does not fit well with our wild packet. The packet we captured was a TCP packet and not an ICMP packet. IP[9] is set to 0x06 which indicates the embedded protocol is TCP. This technique could however be extended to TCP as well as ICMP, so this does not in itself rule out the possibility.

Why make the packet so obvious? There is no need to craft such an ugly packet for this technique to work, all that is needed is to set the reserve bit and know the response that different stacks will give. Unless of course the packet is being crafted to look like it is mangled rather than crafted, or perhaps the use of a fragment is intended to get the packet past some simple packet filtering firewalls.[47] One then needs to ask why an attacker would go to so much trouble. There was only one anomalous packet to this host in the alert logs. Either the attacker is extremely patient or this is not a serious attempt at probing the target host. Nor was there any similar packets to other hosts in the network which would seem to rule out a generic sweep of the network. Perhaps this packet is from a tool under development or some proof of concept code.

Possibility two: Attempted insertion attack

An interesting post to the firewalls mailing list by Cy Ardoin (ardoin@cycon.com)[48] made the point that some kernel code behaves unexpectedly when a packet has the IP reserved bit set. His observation was that the kernel code will "test for ip_offset &~ DONTFRAG but if the reserved bit is set, this test will yield true." What he is saying is that this simple mask presumes the reserved bit is not set. This form of assumption is a possible security problem as many packet filters only filter on the first fragment, which could be fooled should the reserved bit be set and the filtering code assumes otherwise. A malicious user could then insert packets into the network that the firewall is expected to block, as the filtering code would believe them to be

[45] Abnormal may not be an accurate description of this packet, in fact "'Technically, it should be 'Out of Spec' or 'Malformed' packet according to RFC792. " - As pointed out by 'rocker' <starplanet1000@yahoo.com.hk> , in his/her response to my posting this detect on the incidents.org mailing list.

[46] (http://lists.insecure.org/lists/nmap-hackers/2000/Jul-Sep/0068.html)

[47] Many packet filtering firewalls only filter on the first fragment.

[48] http://www.netsys.com/firewalls/firewalls-9610/0570.html

fragments. For this to be the case here, we would expect to see some payload in the packet. There is only 20 bytes of payload however which would require an extremely tightly coded exploit. The payload is not very interesting.

```
                    8329 0050 02fa f904 02fa f90 4    .....).P........
   0x0020 5004 0000 f402 0000 0000 0000 0000       P  ..............
```

There is a repeating pattern of 4 bytes 0x02fa f904 and large section of nulls, but little else of interest. The payload did not trigger snort, nor do we see any evidence of prior reconnaissance, which would surely be needed for such a targeted attack. There is also no evidence of the exploited machine being used by the attacker. This could be simply because the attack failed or perhaps the traffic was not unusual enough for it to trigger the IDS.

Possibility three: packet mangled

A more likely scenario for this packet I feel is that it has been mangled either by a stressed router or some other host on it's route to its destination. There is no other activity related to either the source or destination IP address to indicate malicious activity. There is no real payload in the packet so it is unlikely it is some form of insertion attack. There were only two packets picked up from this IP address that triggered the IDS which indicates it was not reconnaissance activity (unless our hacker is extremely cautious). Also the amount on information that can be gathered by one TCP packet in isolation is relatively low unless the packet was targeted, which I do not believe it was. [49] The data section of the packet is more interesting in the context of a mangled packet than of an 'insertion attack'. It seems more likely that this packet is the victim of a router with a memory problem. The data section seems to include the TCP header from another packet!

If one was to analyse data section as the start of a TPC packet, then the destination port would be port 80, with an ephemeral source port (33577). The sequence number and the acknowledgment number would both be 0x02fa f904 (50002180), and the flags would indicate a Reset packet. Which would make sense as Resets do not usually contain or acknowledge data.

Without more data to look at I would conclude that the most likely analysis is that this is just a 'mangled packet'. I would however consider adding a snort rule to capture data from this address range, or TCP packets with IPIDs of zero to see if any pattern emerged.

6. Correlations:

DShield[50]

Dshield did not report anything on this address

[49] see 7. Evidence of Targeting for details.

[50] Dshield or the Distributed Intrusion Detection System is "an attempt to collect data about cracker activity from all over the intern et. This data will be cataloged and summarized. It can be used to discover trends in activity and prepare better firewall rules. " - http://www.dshield.org

MyNetWatchman[51]

The MyNetWatchman report generated two hits as shown below. It seems that this IP is owned by China Telecom and probably allocated by DHCP (which is common with large ISPs). This means it is highly unlikely that this packet originated from the perpetrator of the alerts listed below. China however is a major contributor to detected attacks, ranking second on the country breakdown at incidents.org when I checked (12/05/2003).

Country	Reports[52]
US	3,182,035
CN	1,693,863
JP	1,074,211
TW	888,999
DE	539,873

Incident Report

Incident Id	Source IP	Provider Domain	Agent Count	Event Count	Incident Status	ISP Resolution Comments
17745271	218.2.129.17	chinanet-js.cn	1	1	Closed	No Recent Activity
5088605	218.2.129.17	chinanet-js.cn	1	6	Closed	No Recent Activity

Incident Detail

Incident Id : 17745271	Source Ip : 218.2.129.17
Provider Domain : chinanet-js.cn	
DNS Name :	
Total Event Count : 1	Total Distinct Agent : 1/0

Most Recent Event Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of IPs Targeted	IP Protocol	Target Port	Port/ Issue Description	Source Port	Explanation	Event Count
2 Jan 2003 18:19:54	Pengwyn	win32	Zone Alarm	10.0.x.x	1	17	137	NETBIOS Name Service W32.Opaserv Worm?	1025	mNW Info	1

7. Evidence of active targeting:

Given that the conclusion I have drawn is that this packet is not malicious the evidence of a targeted attack is moot. However lack of evidence of a targeted attack could help to corroborate my assumption.

As there is no attack signature for this attack no conclusion can be drawn as to whether the attack is targeted to the environment or host in question. Ie if it were an IIS exploit and the target was indeed an IIS host the probability of the attack being targeted is increased.

There is no other supporting traffic in the IDS logs to indicate prior reconnaissance. This would again have increased the chances of the attack being targeted, as a successful targeted attack requires detailed information about the target to succeed.

[51] MyNetwatchMan is a "Security Event Aggregator". It allows multiple firewalls and ids sensors to upload there detects to a central d atabase which can then be used to identify trends, track abusive usage patterns based on IP. It is somewhat similar in concept as Dshield - http://www.mynetwatchman.com/

[52] Internet Storm Center - Country Breakdown - 12/05/2003

8. Severity:

The severity was calculated using the following formula:

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

Each value is to be ranked on a scale from 1 (lowest) to 5 (highest).

Criticality:

There is little we know about the target in question based on these logs files, as there are no other packets, nor indeed any whole packets recorded in the snort logs. This could be simply because no other packets to or form this machines matched a snort signature, or this is the only traffic to or from the host, or the host is simply not there. The latter case assumes that a fragment is being sent to a non existent host. This would be characteristic of a non targeted attack, however there are no other similar packets destined to other hosts, and indeed only one other packet that triggered snort from the same source. Interestingly though this is another fragment, although not such a strange one. It seems even more unlikely that this is the only traffic to or from that host, as this is not even a complete packet!

The first hypothesis seems the most likely, that there was other legitimate traffic between the target and other hosts including the generator of our mystery fragment, but that non of their packets triggered our IDS.

Being a professional paranoid the analyst must in this scenario assume the worst and give the host a criticality of five.

Lethality:

The Lethality of the attack is based on what damage would be done should the attack prove successful. In my analysis I have decided the attack is non malevolent, therefore there is no risk of damage being done to the target system from this packet.

Based on this I give the lethality of this attack a score of one.

System Countermeasures:

Again it is very difficult to attribute a score for system countermeasures from the information of one fragmented IP packet. In this scenario I will again assume the worst and give a score of one.

Network Countermeasures:

From the log files we can see a large number of established sessions inbound from the internet to the protected network. This would indicate that the filtering policy is not extremely tight for this network.

```
$ tcpdump -nr ../2002.5.10.log '(dst net 46.5.00/16) and (tcp[13] & 0x02 !=2)
and (tcp[13] & 0x04 !=4)' | wc  -l
    322
$ tcpdump -nr ../2002.5.10.log '(dst net 46.5.00/16) and (tcp[13] & 0x02 !=2)
and (tcp[13] & 0x04 !=4)' | awk '{print $4}' |sed s: \\.:\ :g |awk '{ print
$5}' | sed s/://g | awk '{ if ($1 < 1024) print $1}' | sort  -nu
   21
```

```
53
80
137
```

This check for well-known ports reveals inbound established connections to HTTP, DNS, FTP, NETBIOS Name Service.

However the network does use IDS at least at its perimeter, which indicates some form of knowledge at least of the need for network counter measures. According to the README file[53] that accompanied this log all ICMP traffic has been removed. This might have been useful in better identifying the filtering policy employed by this network.

Based on all this I will give the Network countermeasures a mid range score of three.

Severity:

Therefore the total score works out to be:
(5+1) - (1+3) = **2**

9. Defensive recommendation:

Seeing as a packet similar to this one could potentially be used in a fingerprinting scan, it may be worth blocking it. This would be a legitimate tactic as technically it is a non-RFC compliant packet.

In addition to this as I suggested in my analysis of the attack, more data could shed light on the true nature of this and similar packets. Adding some snort rules to log packets with an IPID of zero, or from this IP range, for a period of time would be a worthwhile activity.

10. Multi Choice question

The following packet is found in your binary logs.

```
01:41:19.544488 218.2.129.171 > 46.5.188.185:  (frag 0:20@36744)
0x0000 4500 0028 0000 91f1 e706 0178 da02 81ab    E..(.......x....
0x0010 2e05 bcb9 8329 0050 02fa f904 02fa f904    .....).P........
0x0020 5004 0000 f402 0000 0000 0000 0000         P..............
```

Which of the following is the most accurate description?

a) This is a standard IP fragment containing a TCP payload.

b) This is a crafted HTTP packet with a spoofed source address.

c) This is a mangled IP fragment, with no TCP header information.

d) This is a fragment of an packet that contains an HTTP request.

The best answer is **c**.

This is a fragment as can be seen from the friendly tcpdump output. It is also a TCP packet (protocol 6), but it is not the first fragment (offset of 36744) so there is no TCP header here, the 0x0050 in bytes 22:23 is a red herring. It may well be part of an HTTP conversation but we cannot tell from this fragment. It is however mangled, as the IP reserved bit is set. Although it

[53] See Appendix A

could be crafted we cannot be sure from this capture in isolation.

Comments:

Rocker <starplanet1000@yahoo.com.hk>

'Interesting. Is there any other similar analysis in prev GCIA ?'

I could not find any previous GCIA report that covered a similar alert, however "Thomas B. Granier" submitted a detect to the incidents.org mailing list on 27 Nov 2002, which included a number of fragmented packets with the reserved bit set. The detects do not seem to correlate however. In his detect there were over 100 packets in a short time frame. My packet does match up with his patten, as outlined below by Thomas.[54]

```
0x0000 4500 0028  0000 <gggg>  ec06 <chksm>  c001 01bc
0x0010 <dest ip> <xxxx> 0050  <yyyy   yyyy> <yyyy yyyy>
0x0020 0004 0000 <zzzz> 0000   0000   0000   0000
```

<xxxx> is a 2 byte value for which I was unable to determine any pattern.
<gggg> is the offset and IP flags.
<yyyy yyyy> is a 4 byte pattern that is repeated twice.
<zzzz> is a 2 byte value for which I was unable to determine any pattern.

Having a larger number of packets to analyse makes this sort of pattern stand out, and led in Thomas to the conclusion that the packets were generated by a reconnaissance scan tool, designed to evade the IDS. This may well be the case, however a standard install of SNORT includes the 'bad-traffic.rules', which would catch these packets.

Analysis two.

```
[**] [1:477:1] ICMP Source Quench [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/07-12:39:33.894095 81.224.217.33 -> X.X.106.74
ICMP TTL:232 TOS:0x0 ID:45691 IpLen:20 DgmLen:56
Type:4  Code:0  SOURCE QUENCH
```

Posted: https://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00139.html

1. Source of Trace.

This detect was taken from a network I am employed to protect. This gives me a far better understanding of the network topology, and baseline traffic profile. It also allows me to look at the security policy implementation such as filtering rules etc..

[54] I have made the adjustment of inserting a variable for the Fragment offset and IP flags bytes, as the offset in Thomas' packets differed from mine.

The basic layout of the network perimeter is shown in the following illustration, some parts of the network are simplified as they add nothing to the analysis of this attack.

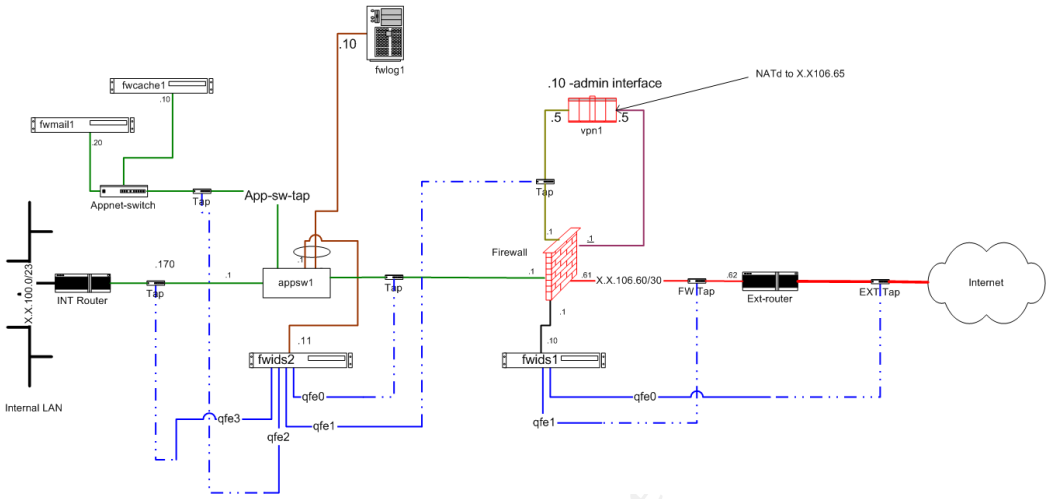


Fig 2

In fig2 above the blue links indicate 'one way' or 'read-only' taps for the two IDS hosts. The detect I am investigating was picked up on the external tap, between our border router and our ISP.

2. Detect was generated by:

Snort Intrusion detection system running a default rule set and preprocessors.

Version 2.0

Rule:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Source
Quench"; itype: 4; icode: 0; classtype:bad-unknown; sid:477;
rev:1;)
```

This rule is intended to alert on ICMP source quench packets entering the network to any internal host. Such packets could be a sign of a DOS attack as we will see.

3. Probability the source address was spoofed:

The triggering packet is a source quench packet from 81.224.217.33, as I will show later in part 4, it is claiming to be in response to a packet destined for port 80 on host 81.224.192.173 from X.X.106.74, the latter being an address allocated to my network. It is not currently being used however, and although the entire /28 block is being routed traffic destined to the unused addresses are currently blocked at the perimeter router. Is this then a legitimate packet replying to a potentially spoofed one, as there is no reason a legitimate packet could have originated from X.X.106.74 ?

The TTL on the ICMP packet is 0xE8 or 232 in decimal. Can we deduce if this is reasonable? A quick traceroute to 81.224.217.33 takes 21 hops to reach its destination. This would indicate an original TTL of 253, so perhaps 255 was the

real TTL as routes on the Internet are often not symmetrical. It would appear then that this packet's credentials check out. Further more 81.224.217.33 does appear to be the last hop on my route to 81.224.192.173.

An active OS fingerprint scan of the host might give us a few last clues.

```
$ nmap -sS -O -osscan-guess 81.224.217.33
Starting nmap 3.28 ( www.insecure.org/nmap/ ) at 2003 -07-10 14:56 XXST
Interesting ports on fls20o1078.telia.com (81.224.217.33):
(The 1634 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     filtered    smtp
53/tcp     open        domain
80/tcp     open        http
1080/tcp   filtered    socks
3128/tcp   filtered    squid -http
4480/tcp   filtered    proxy -plus
6588/tcp   filtered    analogx
8080/tcp   filtered    http -proxy
Remote operating system guess: BSDI BSD/OS 4.0.1 Kernel
Uptime 113.845 days (since Tue Mar 18 18:41:06 2003)
```

So perhaps this is a router or firewall for this network. What else can we learn?

Oh Dear! It looks like it is running an open proxy[55] of some kind.

```
$ telnet 81.224.217.33 80
Trying 81.224.217.33...
Connected to 81.224.217.33.
Escape character is '^]'.
GET http://www.google.com HTTP/1.0

HTTP/1.0 302 Found
Date: Thu, 10 Jul 2003 03:19:55 GMT
Content-Length: 206
Content-Type: text/html
Set-Cookie: PREF=ID=55e938d467798350:CR=1:TM=1057807196:LM=1057807196:S=_5F6mv-svIL3EzU3;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com
Server: GWS/2.1
Location: http://www.google.co.xx/cxfer?c=PREF%3D:TM%3D1057807196:S%3DFE087cBARf_cpkDv
Via: 1.1 nc2-acld (NetCache NetApp/5.3.1R2D4)
<HTML><HEAD><TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A
HREF="http://www.google.co.xx/cxfer?c=PREF%3D:TM%3D1057807196:S%3DFE087cBARf_cpkDv">here</A>.
</BODY></HTML>
Connection closed by foreign host.
```

A 'NetApp NetCache' appliance by the look of it, even Fyodor gets it wrong sometimes I guess.[56] Nice to know these plug and play appliance solutions are so secure by default!

Based on the evidence I do no believe the source address was spoofed.

[55] Open proxies are proxy servers (usuall y HTTP or SOCKS) that allow anyone to make us e of them. They are often used by hackers as a 'connection laundering' proces s, as it makes life harder for the IP detecti ves to track them down.

[56] The NetCache could well run on a BSDI derived OS, but I could find nothing on their web page to support this.

4. Description of attack:

Lets have a closer look at the packet that triggered this alert:

```
2:39:33.894095 81.224.217.33 > X.X.106.74: icmp: source quench
0x0000 4500 0038 b27b 0000 e801 b8cc 51e0 d921   E..8.{......Q..!
0x0010 XXXX 6a4a 0400 72de 0000 0000 4500 0028   .0jJ..r.....E..(
0x0020 0100 0000 6c06 fdc7 XXXX 6a4a 51e0 c0ad    ....l....0jJQ...
0x0030 093d 0050 b93c c757                        .=.P.<.W
```

If we look at the payload of this ICMP packet as with all ICMP error messages, it contains the 'Internet Header + 64 bits of Original Data Datagram'[57] which caused the error condition. Digging it our of the source quench packet we can see the following:

```
4500 0028 0100 0000 6c06 fdc7 XXXX 6a4a       .0jJ..r.....E..(
51e0 c0ad 093d 0050 b93c c757                 ....l....0jJQ...
```

So it is an IPv4 packet with a standard header length of 40 bytes. The transport protocol was TCP, again with a default header length of 40, so no TCP options. The packet was not a fragment as it has a fragment offset of zero and no MF[58] bit set. It had a slightly suspicious ID of 0x0100 or 256. The TTL is 107, which as I indicated earlier makes the originating TTL probably 128, if it was coming from my network. The source address would be X.X.106.74, which is not used. The SEQ number is 0xB93CC757, but we cannot see what the ACK was nor what TCP flags were set.

Looking at the external IDS sensor, which also runs an instance in 'flight recorder' mode monitoring and recording all data, which is kept for 7 days, there is no evidence of any traffic from this network to any addresses in the 81.224.0.0/26 block during this period.

So this packet is either a stimulus packet that is it is not responding to a packet from our network at all, or it is responding to a packet that was spoofing our X.X.106.74 address.

5. Attack mechanism:

The lack of any genuine stimulus packet for this 'Source Quench' message leads me to the conclusion that this alert was the result of a DOS attack of some sort, which utilised source address spoofing to hide the true identity of the attacker(s). One of the addresses spoofed would appear to be the X.X.106.74 address allocated to one of my networks.

A DOS attack is designed to use up all of the victim's resources. In this case network resources were chewed up, as indicated by the router 81.224.217.33 sending the plea for help in the form of the ICMP source quench packet. In order to mask the attackers identity DOS attacks often use source address spoofing. Normally the spoofed addresses would be either unallocated addresses, or idle quiet machines, this has the added benefit of stopping the spoofed hosts from sending RSTs which might help the victim in the case of a

[57] According to RFC792 'Internet Control Message Protocol', in this case we have less to deal with due to the snap-length of the IDS sensor.

[58] More Fragments. - indicates there are more fragments to follow.

SYN flood for example. The /28 address block that this address is in was only allocated in the last couple of months which may explain this.

It is somewhat strange that we only saw one source quench packet, however this was not the only one sent, as indicated by the correlating complaints on the MyNetWatchman site.

6. Correlations:

There was only one GCIA practical I could find that analysed traces of 'source quench' packets. In his GCIA practical, Viriya Upatising concluded his ICMP source quench packets were 'not an attack but a genuine request for the server to slow down the data transfer rate.'

There have been a number of posts on various mailing lists such as Security incidents, where suggestions have been made that the packets are the result of tools such as tcpnice[59]. Which is designed to ' slow down specified TCP connections on a LAN via "active" traffic shaping.' according to it's man page. It does this by either forging a tiny TCP window on outgoing packets or by additionally forging 'ICMP Source Quench' packets.

The MyNetWatcman report for 81.224.217.33, indicates we were not alone in receiving ICMP Source Quench, and interestingly the reports generally tally with ours for the time period, between 0:39 and 1:05 on the 7th (UTC).

Incident Id : 35594359 Source Ip : 81.224.217.33
Provider Domain : telia.com DNS Name : fls20o1078.telia.com
Total Event Count : 10 Total Distinct Agent : 9/9
Response : No Response

Most Recent EventDate/ Time(UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of Ips Targeted	IP Proto	Target Port	Port/Issue Description	Src Port	Event Count
10 Jul 2003 12:12:24	ashram	win32	Zone Alarm	67.85.x.x	1	1	4	ICMP Source Quench	0	2
7 Jul 2003 01:05:45	theserver	Perl	iptables	68.20.x.x	1	1	4	ICMP Source Quench	65535	1
7 Jul 2003 01:00:35	TunaMaxx	win32	Zone Alarm	24.84.x.x	1	1	4	ICMP Source Quench	0	1
7 Jul 2003 00:56:39	wed	Web	Web Form	68.67.x.x	1	1	4	ICMP Source Quench	4	1
7 Jul 2003 00:49:52	Clarke219	win32	Zone Alarm	68.118.x.x	1	1	4	ICMP Source Quench	0	1
7 Jul 2003 00:43:08	LupwaStuff	win32	Zone Alarm	66.75.x.x	1	1	4	ICMP Source Quench	0	1
7 Jul 2003 00:39:18	jhauva	win32	Zone Alarm	200.50.x.x	1	1	4	ICMP Source Quench	0	1
6 Jul 2003 23:40:26	ecc	win32	Zone Alarm	66.9.x.x	1	1	4	ICMP Source	0	1

[59] Part of the Dsniff tool set written by Dug Song.
Another tool put forward was TIDCMP.C proof of concept code by J. Oquendo.

Most Recent EventDate/ Time(UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of Ips Targeted	IP Proto	Target Port	Port/Issue Description	Src Port	Event Count
								Quench		
6 Jul 2003 16:42:58	zippie	Perl	iptables	65.121.x.x	1	1	4	ICMP Source Quench	65535	1

7. Evidence of active targeting:

This traffic was not part of a targeted attack against my network, but was most likely a side effect of a DOS attack either directly on 81.224.192.173, or using this open proxy in a DOS attack on another victim.

8. Severity:

Severity was calculated using the following formula:

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

Criticality:

The packet was destined for an IP address that does not exist so it is hard to identify the criticality of the target host. The address is an Internet routable address intended for use on externally available resources, so as such it would usually be a critical server. Based on these two factors I will give the criticality a two.

Lethality:

The packet was not malicious, but a plea for help from a server under attack somewhere in Sweden. I shall therefore give a score of one for the lethality of this detect.

System Countermeasures.

As the IP address this packet was destined for is not allocated to a host at present, system countermeasures are somewhat moot. All outward viable servers however are hardened before going into production. They are constantly patched to guard against new vulnerabilities, and are run with the guiding principles of 'minimilisation' and least privilege. All hosts in this trust domain also run integrity checking software, have their logs automatically monitored for anomalies, and run a host firewall where possible. Given all this I will give the countermeasures score four.

Network Countermeasures.

Giving a score for the network countermeasures is not easy either. My colleagues and I have recently redesigned the network security infrastructure, but I will try and be impartial and resist giving it a five.

The network uses the principles of defense in depth, and explicit access control. That is there are multiple access control points, and only that which is explicitly permitted is allowed, all other traffic is blocked. This includes egress and ingress

traffic. Very little ingress traffic is allowed at all. Namely return traffic for specified services such as HTTP and DNS, as well as SMTP, and IPSec VPN traffic.

The architecture is comprised of a mix of different vendors, and different technologies. Eg packet filtering, application proxies etc...

The entire system is then backed up with a number of IDS sensors. I will therefore give the network countermeasures a score of four.

Severity = (2+1) - (4+4) = -5

What looked like a promising detect has ended up with a score of -5. Though pleasing for the 'sys-admin' half of my soul, a little disappointing from the 'IDS analyst' half.

9. Defensive recommendation:

As with this network, there is really no need to allow in 'source quench' packets. They could be used in an active DOS attack[60], or to form part of a covert channel. Many trojans are activated by ICMP packets as well, so it is a good idea to limit ICMP in to your network to as little as is needed. In our case that is TIMX packets to some hosts and stateful echo-replies, ie replies to our outbound echo-requests, which are limited to a couple of hosts in this high trust domain.[61]

10. Multiple choice test question:

The following packet is best described by which statement?[62]

```
02:05:15.024880 61.155.14.32 > 192.168.1.1: icmp: source quench (ttl 234, id 18870, len 56)
0x0000 4500 0038 49b6 0000 ea01 dab9 3d9b 0e20   E..8I.......=...
0x0010 c0a8 010a 0400 d57b 0000 0000 4500 003c   .......{....E..<
0x0020 fe2b 0000 ea01 000c c0a8 010 ca66 6198   .+..0........fa.
0x0030 0040 07a9 f91c 255a                       .P....%Z
```

a) It is an ICMP echo request packet, used by the ping program.

b) It is source quench packet from 61.155.14.32 indicating congestion.

c) It is a crafted source quench packet possibly intended as a DOS.

d) It is an IP fragment that has been misinterpreted by tcpdump.

The best answer is c, although the packet is indeed a source quench packet, and not a echo-request nor a fragment, it can't be valid. It is claiming to be in response to a source quench packet from 192.168.1.1 to 202.102.97.152 yet,

An ICMP error message is never generated in re sponse to an ICMP Error message. [63]

Comments: Andrew Rucker Jones

Referring to my portscan of 81.224.192.173:

"Dude, this is so unethical. See the discussion from today about legality, ethics,

[60] A malicious user could spoof an upstream router (such as your ISP's) and force you to stop sending it as much data, effectively strangling your upstream bandwidth.

[61] In other words, no machines on the internal LAN can ping any external hosts.

[62] The checksums will fail as the packet has been obfuscated. The RFC1918 address is also a product of the obfuscation.

[63] Stevens, W Richard - p70

portscanning, and so on."

Response: Interestingly I started the thread Andrew is referring too. I believe that a portscan is a useful tool for the security professional. It should not be abused however, nor capable of damage or excessive network use.[64]

Andrew :

'I am still on the side of blocking everything, including ICMP, if it is not necessary, man y people on that list [OpenBSD] were of the opinion that ICMP is necessary. They took it to the other extreme and wanted to allow all ICMP everywhere, but they had some good points. ICMP is intended to make IP functional. Source quench packets aren't likel y to hurt You, and they may even help You. If i recall correctly (or can believe what i read), sendmail makes use of source quench packets. Although we don't want to make it easy for an attacker to create covert channels, a determined attacker will, and th ere's really no getting around that. All in all, i personally don't see the sense in blocking source quench packets. All that being said, a good stateful firewall will handle source quench packets appropriatel y and reduce the risk associated with them even farther. Explicitly all owing or denying them shouldn't reall y be necessary.'

Response: I agree that allowing in some ICMP to the network can be useful. My argument is to limit the ICMP to that which is truly required. DOS attacks, and covert channels are two obvious reasons uses for ICMP. A number of network mapping techniques make use of ICMP as well. A good stateful firewall should only be one of the components of a secure perimeter. Defense in depth would encourage me to block traffic before it hit my firewall at border routers etc.. Often these are not 'good stateful firewalls' as routers are built primarily to route packet.

Analysis three.

Posted: https://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00141.html

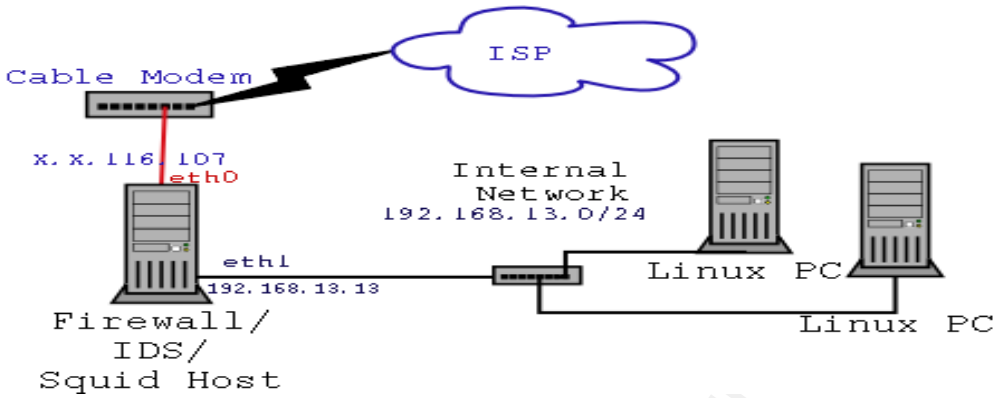
1. Source of Trace.

The following trace was taken from the IDS logs of my home network. This gave me much greater insight into the network layout. As well as a good understanding of baseline traffic behavior. I also have access to traditional log files such as firewall logs, squid logs etc...

The network is very simple consisting of two Linux based PCs, one OpenBSD PC and occasionally a Linux laptop. One of the Linux hosts acts as a server for the other machines. It is the Internet gateway, and also runs a squid proxy. The gateway box runs iptables v1.2.6a and NATs the internal network behind one static Internet address. The internal LAN uses a 192.168.0.0/24 RFC1918 address range. The gateway also runs snort on both interfaces, externally with a full rule set, and internally with a much more limited rule set. The latter mainly as a check that the firewall is up and doing it's job as expected[65], and no unexpected traffic is turning up inside the network.

[64] Are Portscans ill egal? - https://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00161.html

[65] I also run netcat via a cron job to generate a packet that should be blocked by a specific rule in the firewall. If I do not get an alert every X minutes for this, it triggers an alert via logsurfer.



A brief outline of this set up is illustrated above.

Here is the alert that was registered in the snort alert logs from my external sensor.

```
[**] [1:1841:2] WEB-CLIENT javascript URL host spoofing attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
06/15-23:38:31.712634 64.12.152.56:80 -> 192.168.13.13:33095
TCP TTL:62 TOS:0x0 ID:1663 IpLen:20 DgmLen:1500
***A**** Seq: 0x91362E5A  Ack: 0xC6FF921D  Win: 0x4000  TcpLen: 32
TCP Options (3) => NOP NOP TS: 100289073 74105548
[Xref => http://www.securityfocus.com/bid/5293 ]
```

2. Detect was generated by:

The detect was generated by Snort, Version 2.0.0 (Build 72). It was triggered by the following rule:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-
CLIENT javascript URL host spoofing attempt";
flow:to_client,established; content:"javascript \://"; nocase;
classtype:attempted-user; reference:bugtraq,5293; sid:1841; rev:2;)
```

3. Probability the source address was spoofed:

The source address for this detect was almost certainly not spoofed. It was traffic from an established session[66] between the web server 64.12.152.56 and the internal client.

4. Description of attack

bugtraq: 5293 sid: 1841

The alert listed above was triggered at 23:38 on the 15th June, by the external snort IDS sensor.

The source of the offending packet was 64.12.152.56, which resolves to beta-search-vip1.netscape.com. The packet appears to be coming from a webserver (port 80), which matches the modus operandi of this attack, although it also matches the pattern one would expect of a legitimate webserver.

There is no prior traffic to or from this site that triggered the IDS, and

[66] Evidence on this is presented later in the 'Description of attack section'

unfortunately the flight recorders are rolled once every 5 days to save space so I do not have access to these. There is evidence in the squid logs however:

```
1055673509.645 455 192.168.5.254 TCP_MISS/200 16059 GET http://search.netscape.com/nscp_index.adp
- DIRECT/64.12.152.56 text/html
1055673511.891 1067 192.168.5.254 TCP_MISS/200 2144 GET
http://search.netscape.com/images/search_button.gif - DIRECT/64.12.152.56 image/gif
1055673511.212    945 192.168.5.254 TCP_MISS/200 728 GET
http://ar.atwola.com/html/64001910/853209912/aol? - DIRECT/64.12.174.249 application/x-javascript
1055673511.276   1446 192.168.5.254 TCP_MISS/200 329 GET
http://search.netscape.com/images/line.gif - DIRECT/64.12.152.18 image/gif
1055673512.364   1093 192.168.5.254 TCP_MISS/200 3249 GET
```

The site in question is a Netscape search portal just the sort that likes to set cookies[67] for 'site personalisation'. The snort signature that triggered the alarm is intended to alert on attempts to steal cookies from a site other than the site that set the cookie.

5. Attack mechanism:

The problem is that Mozilla allows javascripts to set and read cookies. This does not sound like much of a problem, so long as they are still only allowed to set and read their own. However "for javascript URLs the host and path for the cookie is pulled out as: 'javascript:[host][path]'. Cookie security is based only on restricting access to correct matching host and path. By carefully crafting a mallicious[sic] javascript URL opened in a new frame/iframe/window, it is possible to access and alter cookies from other domains."[68] This bug is listed with Bugtraq and has an ID of 5293. At the time of writing this all Mozilla based browsers prior to v1.01 were vulnerable.[69]

The following example was given by Andreas Sandblad in a post to the Bugtraq mailing list on Jul 24 2002 2:45PM. It demonstrates how to steal a cookie.

```
<body onload=init()>
<iframe name=f height=0 width=0 style=visibility:hidden></iframe>
<script>
function init(){
  f.location = "javascript://www.google.com/\n"+
    "'<body onload=alert(document.cookie)>'";
}
</script>
```

This example will steal, and display the contents of your 'www.google.com' cookie should you have one.

[67] Cookies are chunks of information generated by a web server that is stored on the clients PC. The re are two general types of cookies, that a web server will set. One is a session cookie,these are cookies set by the web server, and stored by the client for the duration of th e session. They are often used by on line stores to keep track of the users shopping cart etc.. The other type of cookie is the persistent cookie. These are a popular means for a server to store information about particular users, often used to personalise sights such as search engines, or on line shoppi ng sites. So that when you request a web page from the server, it reads th e cookie it set last time and knows you are interested in prewar Polish poetry etc... The storing and retrieving of information from cookies generally goes on unnoticed by the user. Importantly cookies are associ ated with a particular server, eg Yahoo.com. Only this server is supposed to be able to read this particular cookie.

[68] Sandblad, Andreas

[69] This includes Netscape 6.2.2 and older.

Why steal a cookie?

There could be many reasons for this; the most obvious is for commercial reasons. The same reason companies like 'double-click' are embedding their cookies in web sites all over the Internet. Companies are desperate to get the edge in selling you things you never knew you wanted. To better know what you need to buy, they want to profile you. Where on the Internet do you go, Snowboarding.com, or philatelists.org? So rather than pay to put their cookie setting code into 3rd party web pages, they could just steal the information.

Another more dangerous use could be to steal session cookies. However these have a limited life span so the attacker would need to know where you are going, and then somehow divert your traffic to there server while the session is still active.

You can see from the following excerpt from the Netscape search engine's HTML page that this alert was triggered by a similar 'javascript://' directive to the example.

```
 <a href="javascript://"
onclick="openWindow('search_tips.html','searchTips','460','420','resizable,scrollbars');
return true;" class=size1>Search Tips</a>
```

I have highlighted the offending javascript in red. In this case though, it is simply opening a javascript window to display the 'search tips' when a user clicks a designated HTML anchor. Not particularly malicious, if a little irritating. There is no server associated with the "javascript://" directive, so it is clearly not trying to pose as another sight. There is also no attempt to read or write a cookie in this segment. It would seem then that this alert is a false positive.

6. Correlations:

I found no previous GCIA practicals that looked into this alert, which was one of the reasons I decided to investigate it. I did find a number of comments on it on the snort users mailing list. Two in particular reflected some of my own thoughts. Both Shane Hickey and Paul Schmehl were of the opinion that the growing number of alerts for this signature in their snort logs were false positives just as I believe mine to be. Paul's alerts were all to the local 'credit union' a site that he knew was frequented regularly by the users at his organisation. They were both critical of the signature, believing it to be written in a manner that leads to the number of false positives.[70]

7. Evidence of active targeting:

This attack is part of a legitimate HTTP conversation between an internal host and a Netscape search engine web server. It is therefore targeted, even if it is not an attack.

The earlier squid log extracts indicate the ongoing HTTP transactions between these two machines, as do a large number of similar entries that are left out for brevity.

[70] I shall expand on this in a little more detail in the Defensive Recommendati ons.

8. Severity:

Severity was calculated using the following formula:

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

Criticality:

The system that was targeted in this attack was a personal workstation. It does however contain some non-replicated data relating to my partners University work. It would therefore be personally devastating for her to have this data lost. As a network security professional it would also be extremely embarrassing, if somewhat educational to have one of my home systems compromised. Not to mention the financial cost of resource theft such as network bandwidth. The system however is not mission critical to any large organisations, so I shall give it a four.

Lethality:

The attack was a false positive, and therefore no damage would result from this particular attack. The lethality of this attack would therefore have to be given a one.

System Countermeasures.

The host system is a Linux host that is patched regularly. It also runs iptables as a form of host protection. I shall only give this system a three, as although it is patched regularly it is used by other users, who have root access.

Network Countermeasures.

The host in question is on a well monitored network. I use snort IDS sensors on the inside and outside of the network monitoring for suspicious network activity. It is also behind a reasonably restrictive firewall. I do not allow any inbound connections what so ever, only return traffic for established connections.[71] I also only allow outbound connections for web traffic from the proxy server, SMTP from the same host, POP3 from all internal hosts, as well as ICMP from all and DNS from all internal hosts. The firewall/server machine is also regularly patched. I shall err on the side of caution and allocate a four .

Severity = (4 + 1) - (4 + 3) = -2

9. Defensive recommendation:

Although this is not a true attack therefore there are no real defensive measures needed, there are a few things that cropped up while researching this alert which could bear some improvement.

Cookies are a useful artifact of the web, but by no means essential. Cookies can be turned off in most web browsers, and indeed Mozilla will allow you to specify sites that are allowed or denied the right to set cookies. This is a feature, which I

[71] This is not quite true anymore as I now allow icmp echo -requests, and TIMX packets in, although I restrict them to 2 per second with a burst max of 5.

now employ, as much to protect my privacy from prying marketers than as a form of defense.

The snort alert rule seems a little easy to trigger for a false positive.

Presuming the stream4 preprocessor[72] has been configured to perform stream reassembly. This ensures the entire javascript function can bee seen by the detection engine, then improving the signature to match something similar to:

content:"javascript\://\w+"; [73]

Would seem a better option. I am using a '\w+' to indicate one or more 'word' characters in this hypothetical rule. [74]

Upgrading your web browser to a version greater than v1.01 if it is a Mozilla variant, and turning off this snort rule would seem the best course of action, especially if you feel your Internet experience would be tarnished by the lack of cookies my first suggestion may impose.

10 Multi Choice Question

The following is best described by which statement?

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-
CLIENT javascript URL host spoofing attempt";
flow:to_client,established; content:"javascript \://"; nocase;
classtype:attempted-user; reference:bugtraq,5293; sid:1841; rev:2;)
```

a) A snort alert generated by a javascript URL similar to:

<a href = "javascript**://**" onclick = "openWindow('search_tips.html'); return true;" Search Tips

b) A snort rule intended to match on web pages with embedded javascript.

c) A snort rule intended to match a javascript URL that is possibly forging it's identity, prior to requesting a clients cookie.

d)A snort rule intended to match an HTTP session with an attempt to write a cookie on the client host.

The best answer is c, although the first option does outline an HTML excerpt that would trigger this rule, it is a rule not an alert. The second option is simply untrue, although javascript is detected by this rule it is a specific use of it, that it is meant to trigger on. Finally option d is way off the mark. The rule is intended to alert the improper access of cookies, it does not trigger on the cookie access but the 'identity forgery'.

Assignment 3 - Analyse this

Assumptions

The protected network of the University was deduced to be MY.NET.0.0/16, which has been obfuscated prior to the logs being analysed.[75] The data used for analysis was incomplete in so much as I often only had snort alerts to work with.

[72] '***preprocessor stream4_reassemble: client port 80*** ' - would be sufficient.

[73] The upcoming 'Regex' feature would be required for this sort of match. (Documented in the 'Writing snort rules' guide at snort.org)

[74] Both Paul Schmehl, and Shane Hickey intimated at this in there posts to the snort users mailing list.

[75] Analysis leads me to believe that this is in fact 130.85.0.0/16 used by the University of Maryland.

Usually no packets or even packet headers. I also had no intimate knowledge of the University's standard traffic patterns; this may have resulted in some systems being identified as suspicious when they may in reality be totally legitimate.

Executive Summary

During the five day period these logs relate to, the University's IDS system captured a total of 2,287,720 scans, 18,596 out of spec packets and 998,821 attack alerts.

There are numerous internal hosts which are running externally accessible services, ranging form peer-to-peer file sharing, web servers, TFTP servers, and IRC servers to name but a few. The University should think very carefully about whether all internal computers should be allowed to host these services, and if these services should be accessible from the Internet. One of the fundamental tenets of network security is 'Deny all except that which is explicitly allowed'. This is particularly pertinent to incoming connections. For example, is there any reason for PCs in the campus dormitories to be running IRC servers that are accessible from the Internet?

Even if a default 'deny incoming connections' policy is not implemented the University's perimeter filtering leaves a great deal to be desired. Incoming netbios, LPD, should simply be blocked unless there are legitimate reasons for the traffic, in which case exception rules should be made. Incoming SNMP should certainly be blocked except for explicit allowed instances.

A great deal of the IDS rules employed by the University are generating more noise than real alerts, and could be easily tightened up. Rules such as the 'High port 65535' for example should only be triggering on SYN packets to internal hosts and the 'Exploit x86 NOOP' alert should be limited to ignore certain ports susceptible to false positives. There is also a great deal of noise from some of the preprocessors which are either out of date such as the defrag preprocessor, or poorly configured such as the 'spp_http_decode' processor.

If the University is to tolerate the use of it's network for peer -to-peer networking or internet gaming it's rule sets should represent this. This could be achieved by ignoring scans from internal hosts to known game ports[76]. This would need to be kept up to date to add new games and prune older games etc... An alternative approach might be to limit external bandwidth to the majority of university hosts (such as dormitory PCs etc..). This would require some traffic shaping but could be a good compromise.

Alternately the IDS could be allowed to continue to trigger as is, but a second layer be introduced such as 'swatch' or 'logsurfer'. The latter with it's statefulness and ability to insert dynamic rules could be very useful. This gives the advantage of still having the alerts for later analysis should it be required.

There is also evidence of spoofed packets being generated on the campus network. If the University's routers are not dropping these then they should be

[76] Unless they match trojan ports.

configured to do so. They should also log the MAC addresses of such packets to allow the offenders to be identified.

Suspicious Internal Hosts

The following hosts were identified during the course of this Analysis as requiring further examination.

Scanners and gamers					
130.85.168.109	130.85.153.187	130.85.137.7	130.85.97.143	130.85.235.110	130.85.202.238
130.85.97.36	130.85.97.233	130.85.97.65	130.85.97.34	130.85.168.177	
130.85.97.172	130.85.97.97	130.85.97. 83	130.85.1.3	130.85.242.250	

nimda		
MY.NET.202.238	MY.NET.97.105	MY.NET.97.97

Services listening on port 65535				
130.85.222.22	130.85.234.190	130.85.249.122	130.85.252.78	130.85.201.58

Tiny fragments		
MY.NET.235.110	MY.NET.250.194	MY.NET.250.50

possible XDCC compromised hosts, or IRC trojans				
MY.NET.112.199	MY.NET.227.246	MY.NET.207.78	130.85.196.193	130.85.226.178
MY.NET.97.122	MY.NET.205.146	MY.NET.97.37	MY.NET.97.43	MY.NET.195.99

Null scans
MY.NET.252.134

In addition to these I detected over 1,000 different hosts actively searching for peer-to-peer ports, be it KaZaa, WinMX, Blublser, Napster, or Gnutella etc... This is eating heavily into the Universities internet bandwidth, as indeed are the two suspected XDCC hosts.

Log Analysis

The University was able to supply me with three different formats of data, for a 5 day period. These are listed below in the table. The files were either alert files, OOS files of scan files. This meant one file per day except for the OOS data for the 08/05 which was supplied in two files.

OOS Files.

The OOS files are Out of Specification packet alerts. That is any packets that do not conform to the normal tcp/ip standards are logged here. These logs include the entire packet rather than just the header as in the case of the alert files. These packets can be very interesting and are often a sign of serious network problems, scanning activity or other malicious activities. Packets used in active fingerprinting are often found here, as they frequently break the RFCs as a means of identifying different OS stacks.

Alert Files.

Each line in the alert files corresponds to a packet that matches one of the snort

rules that the university's IDS is configured to use. Presuming a vanilla install of snort this would equate to approximately 1900 rules. These can be tailored to a sites needs by adding custom rules, and also pruning unrequited rules such as IIS rules for a site that runs solely apache web servers etc... A typical entry from the alert file would look like this:

```
05/07-00:46:13.953360  [**] CS WEBSERVER  - external web traffic [**]
66.196.72.13:40777 -> MY.NET.100.165:80
```

Where the first field is a date/timestamp. This is followed by a delimiter field and then the signature description and a further delimiter. Finally in this example we have a source IP address:port pair, followed by a destination IP address:port pair.

Scan Files.

The scan files are the result of triggers from the snort portscan preprocessor 'spp_portscan.c'.

File List:

Out of Spec	Alert	Scans
OOS_Report_2003_05_08_19128	alert.030507	scans.030507
OOS_Report_2003_05_08_19136	alert.030508	scans.030508
OOS_Report_2003_05_09_1240	alert.030509	scans.030509
OOS_Report_2003_05_10_3171	alert.030510	scans.030510
OOS_Report_2003_05_11_20776	alert.030511	scans.030511
OOS_Report_2003_05_12_28902		

Detects List

The following table lists the attacks caught by the IDS of the five-day period, sorted by type and listed in order of 'hits'. I have also listed the number of different hosts and ports each attack/scan was seen being used against. If the number of hosts or ports was less than three they are listed, this is indicated by an 'H:' or a 'P:' preceding the list, otherwise I have simply listed the total number. The signatures which generated greater than 5,000 alerts are in bold, and will be examined further. These 14 attacks are responsible for over 75% of all the alerts generated over the five-day period.

Attack/Scan Types

Description	Hits	# unique src hosts	# unique dst hosts	# unique dst Ports
Incomplete Packet Fragments Discarded	**317,240**	**112**	**71**	**4**
TCP SRC and DST outside network	**264,573**	**260024**	**168**	**90**
SMB Name Wildcard	**221,089**	**28548**	**41708**	**P: 137, 56464**
High port 65535 udp - possible Red Worm - traffic	**38,044**	**315**	**356**	**46**
Tiny Fragments - Possible Hostile Activity	**25,549**	**19**	**1366**	**P: 56464**
spp_http_decode: IIS Unicode attack detected	**23,157**	**1166**	**1023**	**3**
CS WEBSERVER - external web traffic	**19,038**	**6675**	**H: MY.NET.100.165, 233.2.171.1**	**P: 80, 56464**
High port 65535 tcp - possible Red Worm - traffic	**16,926**	**407**	**169**	**51**

Description	Hits	# unique src hosts	# unique dst hosts	# unique dst Ports
[UMBC NIDS IRC Alert] IRC user /kill detected possible trojan.	**13,097**	**92**	**76**	**6011**
TFTP - Internal TCP connection to external tftp server	**12,045**	**56**	**69**	**79**
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	**9,457**	**6**	**20**	**9**
spp_http_decode: CGI Null Byte attack detected	**8,678**	**222**	**143**	**P: 80, 8080**
Null scan!	**7,393**	**126**	**126**	**342**
EXPLOIT x86 NOOP	**5,387**	**190**	**161**	**113**
TCP SMTP Source Port traffic	2,963	H: 128.32.124.219, 211.147.25.99	2919	3
Queso fingerprint	2,708	405	159	89
MY.NET.30.4 activity	1,617	307	H: MY.NET.30.4	10
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	1,555	11	17	5
IDS552/web-iis_IIS ISAPI Overflow ida nosize	1,114	641	860	P: 80
SUNRPC highport access!	1,014	51	38	P: 32771
connect to 515 from outside	891	3	3	P: 515
CS WEBSERVER - external ftp traffic	865	156	H: MY.NET.100.165	P: 21
Possible trojan server activity	812	66	98	20
MY.NET.30.3 activity	571	45	H: MY.NET.30.3	7
TFTP - Internal UDP connection t o external tftp server	508	36	50	13
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	360	13	8	17
External RPC call	257	4	257	P: 111
NMAP TCP ping!	209	75	83	74
IRC evil - running XDCC	147	11	17	5
SNMP public access	146	15	12	P: 161
EXPLOIT x86 setuid 0	121	115	99	95
EXPLOIT x86 setgid 0	71	65	63	55
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	66	7	63	P: 80
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	57	10	7	12
Notify Brian B. 3.54 tcp	42	37	H: MY.NET.3.54	6
Notify Brian B. 3.56 tcp	40	36	H: MY.NET.3.56	6
Probable NMAP fingerprint attempt	30	14	14	18
SMB C access	30	25	11	P: 139
EXPLOIT x86 stealth noop	24	9	8	8
NIMDA - Attempt to execute cmd from campus host	21	5	21	P: 80
FTP passwd attempt	16	13	H: MY.NET.24.27, MY.NET.24.47	P: 21
SYN-FIN scan!	13	7	6	13
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	7	6	5	7
RFB - Possible WinVNC - 010708-1	3	H: MY.NET.70.225, MY.NET.202.14	H: 68.55.61.117, 24.55.220.133	3
TFTP - External UDP connection to internal tftp server	2	H: MY.NET.202.238	H: 213.64.169.124	P: 4258, 122
[UMBC NIDS IRC Alert] K \:line'd user detected possible trojan.	2	H: 66.252.13.46, 38.115.134.46	H: MY.NET.205.118, MY.NET.201.26	P: 3353, 3509
Attempted Sun RPC high port access	2	H: 129.6.15.29, 216.148.215.98	H: MY.NET.163.23, MY.NET.117.25	P: 32771
DDOS mstream client to handler	2	H: 64.237.37.253	H: MY.NET.205.42	P: 12754
EXPLOIT NTPDX buffer overflow	1	H: 12.129.72.179	H: MY.NET.84.198	P: 123

Description	Hits	# unique src hosts	# unique dst hosts	# unique dst Ports
Back Orifice	1	H: 66.28.238.131	H: MY.NET.163.119	P: 31337
site exec - Possible wu-ftpd exploit - GIAC000623	1	H: 24.186.224.197	H: MY.NET.222.30	P: 21
NETBIOS NT NULL session	1	H: 216.201.238.148	H: MY.NET.132.26	P: 139
Fragmentation Overflow Attack	1	H: 64.109.11.16	H: MY.NET.235.202	P: 0
NIMDA - Attempt to execute root from campus host	1	H: MY.NET.97.105	H: 130.223.20.60	P: 80
DDOS TFN Probe	1	H: 129.41.2.24	H: MY.NET.16.13	

Hosts List[77]

Host	Service	Host	Service	Host	Service
mirrors.umbc.edu	ftp	130.85.249.18	www-http	resnet1-64.resnet.umbc.edu	www-http
linux1.gl.umbc.edu	ssh	130.85.252.251	www-http	resnet4-252-133-r.resnet.umbc.edu	www-http
media.umbc.edu	telnet	resnet2-221.resnet.umbc.edu	www-http	www.umbc.edu	www-http
resnet-gw.umbc.edu telnet	telnet	130.85.130.167	www-http	psc-a.engr.umbc.edu	www-http
media.umbc.edu telnet	telnet	pp1.umbc.edu	www-http	130.85.130.14	www-http
zinc.hhmi.umbc.edu smtp	smtp	resnet2-752.resnet.umbc.edu	www-http	rwd-233.umbc.edu	www-http
resmail.umbc.edu smtp	smtp	130.85.130.34	www-http	rwd-226.umbc.edu	www-http
accessct-server.umbc.edu	smtp	cyclone.umbc.edu	www-http	lab1-05.ifsm.umbc.edu	www-http
mailserver-ng.cs.umbc.edu	smtp	130.85.130.131	www-http	130.85.130.21	www-http
ppp-041.dialup.umbc.edu	smtp	resnet1-150.resnet.umbc.edu	www-http	bio-86-19.pooled.umbc.edu	www-http
resnet2-525.resnet.umbc.edu	smtp	resnet3-46.resnet.umbc.edu	www-http	baltimore.umbc.edu	www-http
asp1.umbc.edu	smtp	130.85.194.245	www-http	ehs.UMBC.EDU	www-http
techport.umbc.edu	smtp	130.85.112.216	www-http	userpages.umbc.edu	www-http
mx2in.umbc.edu	smtp	resnet4-250-122.resnet.umbc.edu	www-http	pplant-80-232.pooled.umbc.edu	www-http
130.85.5.14	smtp	resnet3-437.resnet.umbc.edu	www-http	rwd-237.umbc.edu	www-http
mx3in.umbc.edu	smtp	resnet2-690.resnet.umbc.edu	www-http	rwd-97.umbc.edu	www-http
mx4del.umbc.edu	smtp	130.85.130.86	www-http	resnet1-208.resnet.umbc.edu	www-http
kai.umbc.edu	smtp	chem-87-44.pooled.umbc.edu	www-http	resnet2-362.resnet.umbc.edu	www-http
mx1in.umbc.edu	smtp	130.85.130.64	www-http	lan2.umbc.edu	www-http
ariel2.lib.umbc.edu	smtp	noah.umbc.edu	www-http	linux2.gl.umbc.edu	pop3
mx1del.umbc.edu	smtp	130.85.130.122	www-http	news.umbc.edu	nntp
UMBC3.UMBC.EDU	domain	ndms.umbc.edu	www-http	130.85.190.36	snmp
ecs335pc02.cs.umbc.edu	www-http	ccrf.umbc.edu	www-http	laserjet-304b.umbc.edu	snmp
wt-pubpol-printer.umbc.edu	www-http	130.85.130.91	www-http	wtchem-printer.umbc.edu	snmp
linux3.gl.umbc.edu	www-http	cms.umbc.edu	www-http	130.85.130.200	snmp
bb-app4.umbc.edu	www-	130.85.137.18	www-http	physics205printer.umb	snmp

[77] These Hosts were deemed to be running the following services based on alerts generated and DNS resolution. Some seem expected such as SMTP to mailserver-ng.cs.umbc.edu, others such as snmp to the printers, and the mass of www servers and number of smtp servers perhaps less expected.

Host	Service	Host	Service	Host	Service
	http			c.edu	
wireless-168-218.umbc.edu	www-http	bookstore.umbc.edu	www-http	gestprinter.umbc.edu	snmp
vm-db.umbc.edu	www-http	resnet1-277.resnet.umbc.edu	www-http	resnet3-457.resnet.umbc.edu	synoptics-trap
umbc7.umbc.edu	www-http	130.85.130.27	www-http	130.85.3.56	microsoft-ds
130.85.249.135	www-http	130.85.130.40	www-http	130.85.3.54	microsoft-ds
resnet3-155.resnet.umbc.edu	www-http	project.umbc.edu	www-http	lan1.umbc.edu	microsoft-ds
tfc184.umbc.edu	www-http	centrelearn.umbc.edu	www-http	ecs125xerox.ucs.umbc.edu	printer
resnet1-223.resnet.umbc.edu	www-http	resnet3-113.resnet.umbc.edu	www-http	newprint.umbc.edu	printer
		anubis.cs.UMBC.EDU	www-http	mail.umbc.edu	imap

Networks of interest

The following networks were identified during the course of the analysis:

IP Range	Suspected network	IP Range	Suspected network
130.85.201.0 - 253.0	resnet	130.85.115.0-116.0	biology
130.85.168.0 – 171.0	wireless (154 guest-wireless)	130.85.97.0-98.0	dialup
130.85.166.0 , 100.0, 99.0	Computer Science	130.85.99.0	engineering
130.85.162.0-163.0	physiscs	130.85.90.0	ifsm
130.85.145.0	maths	130.85.130.0	UCS
130.85.140.0	chemistry	130.85.53 55 56.0	ucslab
130.85.138.0	ucslab	130.85.54.0	acslab (Macs)
130.85.136.0	dcs	130.85.7.0	IRC

Most Frequent Alerts

Alert #1 "Incomplete Packet Fragments Discarded"

Snort Rule:-n/a- This message is generated by a preprocessor rather than a snort signature rule. It indicates the university is using the 'frag2' or 'defrag' preprocessor.

Snort SID: n/a **Alerts:** 317,240

Unique Hosts - SRC : 112 **DST :** 71

The frag2 preprocessor was a replacement for the defrag preprocessor, both are used to reassemble packet fragments so that the snort detection engine can be run on the entire packet. They will often trigger when the first fragment is not seen, which can be an indication of an attempted attack, such as an attempt to DOS the IDS, to allow other attacks to take place. Insertion attacks also often use partial fragments.

The majority of alerts were triggered by these two hosts.

Alert Message	Src Host	Src Port	Dst Host	Dst port	Total
Incomplete Packet Fragments Discarded	MY.NET.202.238	0	213.64.169.124	0	316,286

That is 316,286 out of a total of 317,240 alerts, or 99.6% of all alerts. The source and destination port indicated by this alert are seriously suspicious, or are they? Remember that these fragments have triggered the preprocessor to

alert because they were incomplete, so we can assume that the TCP header was missing hence the lack of port information.

So who is our mystery external host and why is MY.NET.202.238 sending him all these fragments that are triggering the preprocessor?

First off lets look at the external host.[78]

```
# nslookup  213.64.169.124
h124n2fls33o812.telia.com
```

So this host is another RIPE address, this time it is situated somewhere in the Netherlands.

A quick check on MyNetWatchman reveals the following:[79]

Most Recent Event Date	Agent Alias	Agent Type	Log Type	Target Ip	# of Ips	Proto	Target Port	Port/Issue Dsc.	Src	Event Count
12 May 2003 08:19:26	jankemi	Perl	Cisco PIX	134.29.x.x	12	6	80	HTTP Probable CodeRed/Nimda	1483	34
11 May 2003 18:37:28	Computer	SOAP/XML	Zone Alarm	10.0.x.x	2	6	80	HTTP Probable CodeRed/Nimda	5034	2
10 May 2003 22:12:16	Atlas	win32	Zone Alarm	213.64.x.x	2	6	80	HTTP Probable CodeRed/Nimda	18296	6
10 May 2003 17:18:52	Unspecified	win32	Zone Alarm	24.83.x.x	1	6	80	HTTP Probable CodeRed/Nimda	53455	3
8 May 2003 21:39:28	aclark	win32	Zone Alarm	192.168.x.x	1	6	80	HTTP Probable CodeRed/Nimda	10866	2

These two hosts were also involved in the following alerts:

Alert Message	Source IP	Src port	Dst IP	Dst port	totals
High port 65535 udp - possible Red Worm - traffic	MY.NET.235.102	0	213.64.169.124	0	1
TFTP - External UDP connection to internal tftp server	MY.NET.202.238	69	213.64.169.124	122	2
High port 65535 udp - possible Red Worm - traffic	MY.NET.202.238	3369	213.64.169.124	65535	8

There were also some SMB wildcard alerts triggered for the internal host from a number of other hosts, but I believe that this is just noise.

The first alert for the incomplete fragments is at 05/10-20:30:01 with the last being at 05/11-23:54:01 a period of 26 and one half hours. So although there were over 316,000 packets sent they were spaced out over a fairly large period, not the usual pattern for a DOS attack.

Also in a post on the snort users mailing list Marty Roesch had the following to say regarding a similar output.

> "That means that you're using the defrag preprocessor instead of the newer frag2 preprocessor and that you should switch to frag2. :) The defrag preprocessor had some fairly nasty failure modes and has since been superceded[sic] by frag2, so I'd recommend using that for now."[80]

MY.NET.202.238 host may also be running a TFTP server, which is accessible from the outside world, or perhaps more likely based on the MyNetWatchman data is infected with nimda.[81] Interestingly there have been 30 reports listed

[78] For more information see the External Address Registrations Section.
[79] MyNetWatchman - http://www.mynetwatchman.com/LID.asp?IID=30378477
[80] Roesch, Marty
[81] Nimda often propagates itself in part via tftp copying the file 'Admin.dll' from an infected to host to a

against this IP at MyNetWatchman. The details are no longer available, presumably because the host has been cleaned.

Recommendation: This could well be a false positive. The university should check if it is indeed using the latest preprocessor, if not an upgrade is highly recommended as the noise this alert is generating is masking other more interesting traffic. If this is not the case then the MY.NET.202.238 host should be examined for possible contamination or some form of network problem. I think this is more likely else we would see the alert trigger for more hosts. Also incoming TFTP traffic should be blocked at the perimeter, unless it is explicitly required, in which case it should be restricted as much as possible. MY.NET.202.238 should be examined as a matter of course for possible nimda infection.

Alert #2 "TCP SRC and DST outside network"

Snort Rule: n/a

Snort SID: n/a **Alerts:** 264,573

Unique Hosts - SRC : 260,024 **DST:** 168

Over a quarter of a million triggers for this alert!! What is going on?

The first thing one notices is that there are nearly as many different source addresses as there are host addresses. The great majority are in the 90.0.0.0/8 address range, nearly 260,000 alerts.

My initial thought was this is an automated scan of some sort, sweeping whole segments. But this does not fit as neither addresses are in our network, so either we have a serious routing issue, or more likely one of the addresses is spoofed.

Ignoring the source addresses which are not in the 90.0.0.0/8 block the first trigger we see was at '05/08-05:22:31.129328' to 216.74.66.94:6667, and the last packet is at 05/08-05:28:48.219065'. That is a total elapsed time of six minutes and17 seconds. During this short period this one host received 151,890 packets! That is an average of over 400 packets per second! Could this be a DOS[82] attack, or maybe a DDOS[83] ?

A quick look at my /etc/services file reveals the following:

```
# grep 6667   /etc/services
 ircd   6667/tcp    # Int ernet Relay Chat
 ircd   6667/udp    # Inter net Relay Chat
```

Now that is interesting. The traffic was directed at an IRC[84] server. In fact this is Luna.Elite-Irc.Net, which can be seen by connecting to port 6667 on this IP:

```
Welcome to the Elite-Irc IRC Network scouser!~scouser@X.X.X.X (from Luna.Elite-Irc.Net)
Your host is Luna.Elite-Irc.Net, running version Unreal3.2-beta17 (from Luna.Elite-Irc.Net)
This server was created Sat Jun 21 2003 at 22:05:13 EDT (from Luna.Elite-Irc.Net)
Luna.Elite-Irc.Net Unreal3.2-beta17 iowghraAsORVSxNCWqBzvdHtGpD
```

new victim.. - Schmelzel, Paul

[82] DOS - Denial of Service Attack.

[83] DDOS - Distributed Denial of Service Attack.

[84] IRC - Internet Relay Chat, a real time communication system often used by hackers, for sharing information, and lau nching DDOS attacks.

The target here is a host in the Interserver Inc address space, according to a quick search using Sam Spade.org.

```
Virtual Development INC VDI-1-BL (NET-216-74-64-0-1)
                                    216.74.64.0 - 216.74.127.255
Interserver, Inc INTERSERVER-216-74-66-0-24 (NET-216-74-66-0-1)
                                    216.74.66.0 -  216.74.66.255
```

Next we see a similar pattern, only this time it is from addresses in the 94.66.0.0 - 94.93.0.0. range, and the target is now 62.67.226.90:90. That equates to 111,717 packets in two minutes and 37 seconds.

This time I had to consult IANA to find something out about TCP port 90:

```
#### PORT 90 also being used unofficially by Pointcast #####
   dnsix 90/tcp DNSIX[85] Securit Attribute Token Map
```

Pointcast seems the more likely here. The target is not a US military or government IP address, but in fact a European address owned by RIPE.[86]

A traceroute to this address indicates that it is in Germany somewhere near Dusseldorf.

```
....
12  ge-9-0-0.core1.dus1.de.inetbone.net (62.67.36.186) 304.651 ms  303.956 ms  305.317 ms
13  ge-0-0.customer1.dus1.de.inetbone.net (213.203.192.230) 302.312 ms  301.956 ms  302.554
ms
14  62.67.226.90 (62.67.226.90)  303.762 ms  302.910 ms  303.127 ms
```

So what is going on here?

Firstly it seems highly probable that all the 90.0.0.0/8 addresses are spoofed in order to hide the true source of the packets. This could be easily established if we had access to some packet captures. One would hope the universities routers would simply drop these packets, as they have no reason to be on the network. The majority of the packets come in large bursts to the two targets identified above. This seems to fit in with some sort of DOS attack. The first target is possibly an IRC server. IRC is a common tool of the DDOS, used to martial armies of contaminated hosts. These hosts are infected usually with a trojan which will log on to a specific IRC channel once activated and wait for commands. 400pps is a lot of packets especially if they are large packets, but it is not an unbelievable amount[87]. Perhaps then this is part of a DDOS attack and we are only seeing a small fragment of it, other 'zombies' could well be contributing from other networks.

Recommendation: There is enough evidence here to indicate possible compromised hosts on the network. Analysis of IDS packet traces could be used to identify which hosts are generating these packets if they have been kept, these hosts can then be examined. If not I suggest some form of archiving of IDS packet dump data, so future problems will be easier to solve, and continued monitoring of this type of traffic.

---

[85] Defense Intelligence Security Information Exchange -http://www.fas.org/news/reference/terms/d.html

[86] RIPE (Réseaux IP Européens): A consortium of " Regional Internet Registries that exist in the world today, providing allocation and registration services that support the operation of the Internet globally. ... primarily for the benefit of the membership in Europe, the Middle East, Central Asia and African countries located north of the equator."

[87] I managed to generate over 800pps from my test PC while transferring very large files over a 100Mb link.

Alert #3 "SMB Name Wildcard"

Snort Rule:

alert UDP $EXTERNAL any ->; $INTERNAL 137 (msg: "IDS177/netbios_netbios-name-query"; content: "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00 00|"; classtype: info-attempt; reference: arachnids,177;)[88]

Snort SID: arachnids 177 **Alerts:** 221,089

Unique Hosts - SRC : 28,548 **DST:** 41,708

According to whitehats.org, "This event indicates a standard netbios name table retrieval query.". So this could well be legitimate traffic. It is used as part of the file sharing protocol for name resolution, however it could well provide, host names, domain names, and logged on user details to a would be attacker.

There were over 220,000 separate triggers for this event, which could well be normal background noise of a busy University campus. However the traffic is almost entirely incoming to hosts on the University network from the Internet. This indicates attempts to access file shares from external networks. This is a serious concern. As of 27 June 2003 the incidents.org website was listing the netbios-ns port (137) as the most attacked port on the Internet. It goes on to list eight vulnerabilities for this port from CVE.[89]

Recommendation: This incoming traffic should be blocked at the border routers/firewalls.

Alert #4 "High port 65535 udp - possible Red Worm– traffic" & Alert #8 "High port 65535 tcp - possible Red Worm – traffic"

Snort Rule: n/a - custom rules

Snort SID: n/a **Alerts:** #4: 38,044 #8: 16,926

Unique Hosts - SRC: #4: 315 #8: 407 **DST:** #4: 356 #8: 169

Red worm, or 'Adore' worm is a collection of programs and scripts, which attempt to gain unauthorized access to systems running LPRng, rpc-statd, and the Berkeley Internet Name Domain (BIND).

Should 'Adore' find a vulnerable system it will install a trojaned version of 'ps' and the wait for a control message. This comes in the form of a crafted icmp packet. Once the control packet arrives 'Adore' opens a back door on the system listening on TCP port 65535. Once infected it would classically attempt to 'transmit data identifying the compromised systems to four different e-mail addresses, two of which are in China and two located in the U.S.'[90]

The worm will also randomly generate the first 2 octets of an IP address and then scan that entire subnet range for any other vulnerable systems.

[88] I could find no alert with the "SMB Name Wildcard" signature message, however the net bios-name-query signature which looks for the infamous 'CKAAA..' payload above sourced from whitehats.org would would alert on this.

[89] See Appendices for a listing.

[90] McDonald, Tim

According to J Anthony Dell[91] however the worm uses TCP not UDP for its back door, so we should be able to ignore the alerts for the port 65535 UDP activity.

Port 65535 is a legitimate port used by hosts as an ephemeral port, so an important question has to be if these alerts were due to stimulus packets or responses. If the packets were responses then the behavior is just normal network traffic, however if they were stimulus packets then we should be suspicious. This would be easier to tell with the TCP alert as it is a stateful protocol, so we could assume the signature would only trigger on SYN packets. This could account for the lower number of triggers.

For these alerts to be of concern we should see connections to port 65535 on a 'MY.NET' host, preferably with some corroborative evidence, either large scans, or emails emanating form this host. There may also be evidence of prior reconnaissance although this could well be in earlier logs.

Using this as a guide I get six candidates from the TCP alerts and 15 from the UDP alerts.

Cross referencing for scanning activity gives me the following shortlist.

Source IP	Initial Alert	Scans
130.85.222.22	UDP	13
130.85.234.190	UDP	341
130.85.249.122	TCP	1,156
130.85.252.78	TCP	1,485
130.85.201.58	UDP	1,856

The scanning activity of these hosts does not match the expected behavior of an 'Adore' infected host, as none of the scans are for the services which this worm targets.

Looks like a false alarm, however it would be interesting to know what programs these machines are running bound to port 65535, particularly the two TCP triggering hosts.

Recommendation: The rules that are attempting to catch Adore should be tightened, to only trigger on attempts to establish a connection to port 65535 TCP on an internal host. Further to this I recommend blocking un-established incoming traffic to port 65535 TCP.

Alert #5 "Tiny Fragments - Possible Hostile Activity"

Snort Rule: n/a - minfrag preprocessor

Snort SID: n/a **Alerts:** 25,549

Unique Hosts - SRC**:** 19 **DST:** 1,366

The minfrag preprocessor will generate an alert on any fragment that is smaller than a set size. It operates on the assumption that modern network hardware has no reason to fragment a packet smaller than a certain size. So a fragment that is below this threshold value is a possible indication of an attacker trying to slip a packet in through your perimeter defenses. The packets can be so small as to have information such as port numbers in subsequent packets. The fragment offsets can be such that it overlaps a previous fragment thereby

[91] Dell, J Anthony

overwriting the port number with the attackers true target port.

The majority of the alerts were triggered by traffic from MY.NET.235.110 to hosts on the internet. In fact this host generated 25,149 alerts out of a total of 25,549. This host also triggered 2,857 alerts in the scan logs to a variety of different external hosts.

Is this an attack then or something else, peer-to-peer file sharing applications such as Gnutella have been known to generate false positive.[92] There is some evidence that this person uses KaZaa and Gnutella from their scanning activities.

The inbound traffic that triggered this alert is probably of more concern.

Source IP	Destination IP	total	Source IP	Destination IP	total
68.212.64.248	MY.NET.250.50	151	141.156.193.216	MY.NET.217.222	1
68.212.64.248	MY.NET.250.194	91	82.65.127.218	MY.NET.226.178	1
213.23.14.81	MY.NET.210.82	83	65.71.58.229	MY.NET.209.206	1
4.47.132.210	MY.NET.250.194	24	64.113.65.83	MY.NET.229.126	1
209.50.91.146	MY.NET.222.118	16	219.53.0.47	MY.NET.235.78	1
4.33.2.230	MY.NET.235.86	14	218.54.64 .26	MY.NET.204.26	1
219.53.0.47	MY.NET.250.78	4	220.85.119.246	MY.NET.235.74	1
61.102.204.119	MY.NET.217.6	4	218.15.242.31	MY.NET.100.10	1
61.146.216.98	MY.NET.100.10	1	12.231.152.232	MY.NET.219.242	1

Looking for other alerts generated by these addresses shows a great deal of interest in a number of hosts on the University network from 68.212.64.248 .

Alert Message	Source	Target	Hits
Queso fingerprint	68.212.64.248	MY.NET.250.50	1
Tiny Fragments - Possible Hostile Activity	68.212.64.248	MY.NET.250.50	151
Null scan!	68.212.64.248	MY.NET.250.50	112
Tiny Fragments - Possible Hostile Activity	68.212.64.248	MY.NET.250. 194	91
Null scan!	68.212.64.248	MY.NET.250.194	72
Queso fingerprint	68.212.64.248	MY.NET.250.194	1
Probable NMAP fingerprint attempt	68.212.64.248	MY.NET.250.194	1
Tiny Fragments - Possible Hostile Activity	68.212.64.248	-	1

There are also similar entries in the scan logs. So who is our curious friend?

```
$ nslookup  68.212.64.248
 name = adsl-212-64-248.chs.bellsouth.net
```

So another broadband DSL user. No useful information on Dshield or MyNetWatchman, but chances are this is a dynamically allocated IP anyway.

There is enough indication that this is information gathering on the part of 68.212.64.248 to warrant further monitoring.[93]

Recommendation: The hosts MY.NET.235.110, MY.NET.250.194, MY.NET.250.50 should all be examined for possible compromise. The latter two are both listed with MyNetWatchman.

Alert #6 "spp_http_decode: IIS Unicode attack detected"

Snort Rule: n/a -http_decode preprocessor
Snort SID: **Alerts:** 23,157
Unique Hosts - SRC: 1,166 **DST:** 1,023

This alert is triggered by the snort preprocessor when it finds Unicode encoded

[92] Neohapsis, snort mailing list archive - "Tiny Fragments", Laurie Zirkle
[93] Registration details for this host are included in the 'Five External Hosts' section

characters[94] in an HTTP data stream. Many evasion techniques make use of Unicode, by encoding the characters or patterns that an IDS, firewall or application may block or reject. The characters still have the same meaning but are subtly disguised. In addition a number of applications have had problems with the way they un-encode Unicode characters. It is a technique used for evasion by tools such as whisker[95], and for attack purposes by worms such as Code Red and Nimda.

This signature has been marked as one that generates a large number of false positives, especially if the traffic is having to deal with unusual character sets such as Korean or Japanese etc..[96]

Some however do not appear totally innocent.

First we see our Unicode type attacks to these hosts:

TIME	Alert message	Src IP	Dst IP	hits
05/10/03 23:50	IDS552/web-iis_IIS ISAPI Overflow ida nosize	80.17.44.90	**MY.NET.97.105**	1
05/07/03 22:01	IDS552/web-iis_IIS ISAPI Overflow ida nosize	61.171.133.120	**MY.NET.97.97**	1

Next some attempts to run cdm.exe or root to other hosts

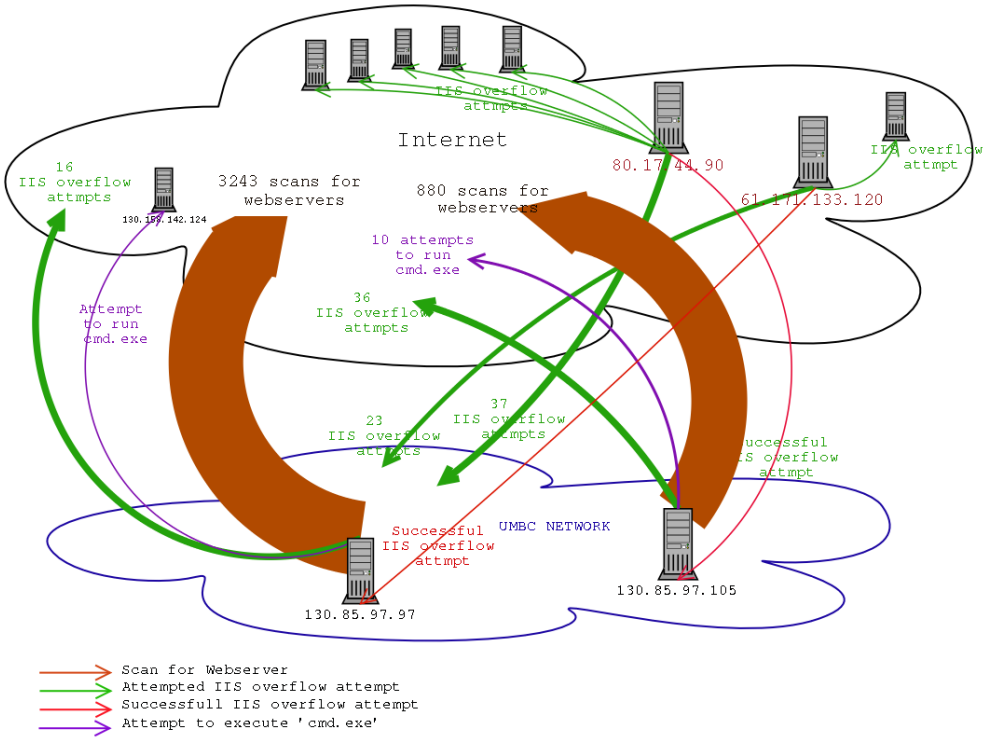
05/10/03 23:50	NIMDA - Attempt to execute root from campus host	MY.NET.97.105	130.223.20.60	1
05/11/03 00:17	NIMDA - Attempt to execute cmd from campus host	MY.NET.97.105	130.94.230.29	8
05/11/03 21:36	NIMDA - Attempt to execute cmd from campus host	MY.NET.97.97	130.158.142.124	1

We also see 1205 scans for network shares from MY.NET.97.97 between 12:47:20 and 13:11:15 on the 8th May. This is nearly 24 hours after the probable infection. On the 11th at 21:30:03 we then see a scan for web servers start, which triggers 2,164 alerts.

Checking my two candidates for the source of infection with MyNetWatchman gave me the following:[97]

Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of target Ips	IP Proto	Target Port	Port/Issue Description	Src Port	Exp.	Event Count
1 Jul 2003 13:45:56	jankemi	Perl	Cisco PIX	134.29.x.x	17	6	HTTP	Probable CodeRed/Nimda	1166	mNW Info	51
26 Jun 2003 09:44:50	gibosi	win32	Zone Alarm	128.146.x.x	1	6	HTTP	Probable CodeRed/Nimda	3315	mNW Info	1
24 Jun 2003 19:38:18	nullbob	Perl	Cisco PIX	209.176.x.x	1	6	HTTP	Probable CodeRed/Nimda	2416	mNW Info	3
24 Jun 2003 00:11:02	ajg	win32	Zone Alarm	128.120.x.x	1	6	HTTP	Probable CodeRed/Nimda	4549	mNW Info	1
18 Jun 2003 01:03:22	joero9	win32	Zone Alarm	128.118.x.x	1	6	HTTP	Probable CodeRed/Nimda	4089	mNW Info	1
61.171.133.120											
15 May 2003	jankemi	Perl	Cisco PIX	134.29.x.x	14	6	HTTP	Probable CodeRed/Nimda	3099	mNW Info	257

[94] Unicode is a two-byte encoding method which covers all of the world's common writing systems.
[95] An IDS evasion tool developed by Rain Forest Puppy.
[96] Gordon, Les M & Carpenter Carlin.
[97] The time stamp given is for the latest instance, and it appears theses machines are still infected with Nimda! (1 July 2003)



Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of target Ips	IP Proto	Target Port	Port/Issue Description	Src Port	Exp.	Event Count
22:29:29											

The following Link graph outlines the probable infection vector and subsequent activity of the two hosts MY.NET.97.105, and MY.NET.97.97.

Recommendation: I recommend the University use the '--unicode' option for this preprocessor to cut down on noise, or even perhaps use a bpf filter to ignore outbound traffic to port 80, as it looks like this filter has triggered on at least two legitimate cases. The two hosts identified above are almost certainly infected with Nimda and should be examined.

Alert #7 "CS WEBSERVER - external web traffic"

Snort Rule: n/a custom rule
Snort SID: n/a **Alerts:** 19,038
Unique Hosts - SRC: 6,675 **DST:** 2

It seems likely that this is a custom rule that triggers when external hosts attempt to access the 'CS Web server', which seems most likely to be MY.NET.100.165. There are two instances of this rule triggering on destination IP 233.2.171.1, which are I believe are errors, or corruption. It would follow then that external hosts are not intended to be able to access this web server. It may host sensitive research data etc.. It does appear to have been scanned but only

as part of a general sweep rather than a targeted scan.

Recommendation: Test the integrity of the filtering of web requests to this host, possibly also introduce some host based filtering such as ipfilter, or iptables for a Unix host, or tcpwrappers[98] etc.. Analysis of the logs should tell us if these requests are actually being serviced or not. Filtering these requests at the border would reduce the amount of noise in the IDS reports, and provide an additional layer of security for this server.

Alert #9 "[UMBC NIDS IRC Alert] IRC user /kill detected possible trojan."

Snort Rule: alert tcp $EXTERNAL_NET 6660:7000 ->; $HOME_NET any\ (content: "ERROR \:Closing Link\: "; nocase;\ flow: established;\ msg: "IRC user /kill detected, possible trojan.";\ classtype:misc-activity;)[99]

Snort SID: n/a

Alerts: 13,097

Unique Hosts - SRC: 92 **DST:** 76

The /kill command is used to cause a client-server connection to be closed by the server; it is sometimes available to users with 'operator' status. It breaks the flow of data and can be used to stop large amounts of 'flooding' from abusive users. It is often issued automatically by IRC servers when a user attempts to login with a nickname that is already in use.[100]

This alert then is triggered whenever an 'ERROR \:Closing Link\:' message is issued from and external IRC server (specified in this case to be 6660:7000) to an internal client. Presumably the alert is intended to identify internal hosts infected with trojans that use IRC channels as a form of communication. There are not many legitimate reasons for a client to be /kill'd.

Of the internal hosts that triggered this alert, the following triggered ' IRC evil - running XDCC[101] ' also.

Alert Message	Source IP	total
IRC evil - running XDCC	MY.NET.112.199	3
IRC evil - running XDCC	MY.NET.227.246	5
IRC evil - running XDCC	MY.NET.207.78	57

And the following also appeared in the scan logs.

Source IP	Destination IP	Dst port	Scans
130.85.194.131	192.168.19.7	4665	21
130.85.238.158	213.65.128.83	65199	23
130.85.235.102	66.151.181.39	7777	29
130.85.226.178	61.122.212.188	6257	1,947
130.85.196.193	24.0.51.70	17300	1,341,125

Interestingly 130.85.196.193 is an extremely noisy scanner looking for hosts

[98] Wietse Venema's network logger, also known as TCPD or LOG_TCP. Allows monitoring and filtering of access to network services such as FTP, Telnet etc..

[99] Courtesy of Nick Nelson < nick@arpa.com > - http://arpa.com .

[100] Kalt, C - RFC 2812.

[101] XDCC is 'eXtended Direct Client Communication protocol'. According to the RFC - 'It's primary. purpose allows the client to act as a file server, being automatically able to initiate DCC SEND requests'.

listening on port 17300. I investigate this further in my analysis of the 'Top Talkers'[102].

Recommendation: It is very difficult to identify if this is trojan activity. It may be easier therefore to restrict the hosting of internal IRC servers to either none, or a known group. Then block IRC traffic to all but these hosts either by filtering the usual IRC ports or all incoming connection attempts. If the IRC servers list is not empty then these hosts should be carefully monitored, and kept up to date with regard to patches etc.. It would be nice if these servers could also be quarantined from the rest of the University Network, as un-trusted hosts.

Investigate MY.NET.112.199, MY.NET.227.246, MY.NET.207.78, 130.85.196.193, 130.85.226.178, for evidence of compromise. The first three are possibly being used as 'warez' servers for the distribution of illegal software, movies, mp3s etc.. This is usually accomplished by compromising 'windows' PCs due to externally accessible file shares and week Administrator passwords. Incoming netbios traffic should be blocked at the perimeter, and the University should ensure all campus hosts have adequate passwords.[103]

Alert #10 "TFTP - Internal TCP connection to external tftp server"

Snort Rule: n/a custom rule

Snort SID: n/a **Alerts:** 12,045

Unique Hosts - SRC: 56 **DST:** 69

This is a custom rule that is designed to trigger whenever a host in $HOME_NET connects to port 69 on an external host, ie one not within $HOME_NET.

This could be a sign of a vulnerable web server transferring the nimda worm in the form of 'admin.dll' from a previously infected server. There is no corroborative evidence for this however such as any of these hosts triggering 'Attempt to execute cmd.exe from campus host' or mass scanning for web servers etc..

There is a great deal of traffic from 66.42.68.210 to MY.NET.201.58 which seems to be listening on UDP port 65535.

Alert Message	Src IP	Src port	Dst IP	Dst port	Total
High port 65535 udp - possible Red Worm - traffic	MY.NET.201.58	65535	66.42.68.210	5122	42
High port 65535 udp - possible Red Worm - traffic	66.42.68.210	5121	MY.NET.201.58	65535	10,550
High port 65535 udp - possible Red Worm - traffic	MY.NET.201.58	65535	66.42.68.210	5121	11,001

Previous scanning activity by MY.NET.201.58 host.

Protocol	Source IP	Destination IP	Target port	Total
UDP	MY.NET.201.58	66.42.68.210	5122	18
UDP	MY.NET.201.58	66.42.68.210	5121	58

[102] Aka 'Mr Noisy' - Top talkers.

[103] Password auditing programs such as lophtcrack etc.. could be used by the campus administrators.

This makes me think that 201.58 is looking for a NeverWinter Nights server[104].

Recommendation: If there is a legitimate reason for some hosts to connect to external TFTP servers then there should be explicit exceptions placed into the snort config, or custom rules file. Otherwise outgoing TFTP connections should be blocked at the perimeter.

Alert #11 "[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC"

Snort Rule: n/a - custom rule

Snort SID: n/a **Alerts:** 9,457

Unique Hosts - SRC: 6 **DST:** 20

This rule would appear to be a custom rule similar to the one used in alert #9. It is difficult to deduce how accurate it is without being able to look at the rule, as such I will assume no false positives.

The sdbot and floodnet are trojans that give back door access to the victims machine via IRC. The trojan contacts an IRC channel via a built in IRC client and waits for instructions. It would seem likely that it is these commands that along with the IRC channel on a connection to port 6660:7000 that the snort signatures look for. [105]

There are only five internal hosts that are triggering this signature for outbound connections:

Source IP	Hits
MY.NET.97.122	1
MY.NET.205.146	1
MY.NET.97.37	1
MY.NET.97.43	1
MY.NET.195.99	6741

The majority of the alerts are from 195.99, which has also triggered alerts for ' IRC evil - running XDCC '.[106]

Recommendation: Investigate the five hosts listed above for evidence of compromise. Blocking IRC from the University would be nice, but hard to police. No doubt it would also prove highly unpopular.

Alert #12 "spp_http_decode: CGI Null Byte attack detected"

Snort Rule: n/a -preprocessor

Snort SID:

Alerts: 38,678

Unique Hosts - SRC: 222 **DST:** 143

This is an alert triggered by the spp_http_decode preprocessor indicating it has seen an instance of '%00' (an escaped Null) in the data. This is attempting to catch malicious input to cgi scripts notably those written in perl.

The exploit is possible because perl unlike a lot of languages (importantly

[104] A popular multi-player Role playing game.
[105] Symantic Security Response
[106] See Alert #9 "[UMBC NIDS IRC Alert] IRC user /kill detected possible trojan." for details.

including C as we will see later) allows the null character as part of a variable, and does not treat it as a string delimiter. In the words of RFP[107] as far as perl is concerned:

> *"root" != "root\0". But, the underlying system/kernel calls are progr ammed in C, which DOES recognize NUL as a delimiter.*

So in the above example a check for 'root' will fail in perl, but the system call will only see 'root'! As with many content-based signatures this alert is prone to a great deal of false positives.

Recommendation: The best defense for this form of attack is good coding practices for CGI code on the University's web servers, especially any that are accessible from the Internet. Un-tainting all user input data should be standard practice, escaping meta characters such as `\"|*?~<>^()[]{}$\n\r and removing any NULL characters completely. I would suggest, as with the IIS Unicode alerts, disabling this feature with the '-cginull' option to the spp_http_decode preprocessor configuration line.

Alert #13 "Null scan!"

Snort Rule: alert tcp $EXTERNAL_NET any ->; $HOME_NET any (msg:"SCAN NULL"; flags:0; seq:0; ack:0; reference:arachnids,4; classtype:attempted-recon; sid:623; rev:1;)

Snort SID: 623

Alerts: 7,393

Unique Hosts - SRC: 126 **DST:** 126

A Null scan is a TCP scan where the attacker sends a TCP packet with a sequence number and acknowledgment number both set to 0, as well as all the control bits set to 0, that is no flags set. The following packet was generated by one of my lab hosts running the nmap tool[108] and then sniffing the wire using tcpdump.

```
#nmap -sN 203.96.152.15
myhost.63295 > 203.96.152.15.26: . [tcp sum ok] win 3072 (ttl 50,
id 14707, len 40)
4500 0028 3973 0000 3206 ac32 xxxx xxxx
cb60 980f f73f 001a 0000 0000 0000 0000
5000 0c00 0960 0000
```

As you can see from this dump none of the TCP flags are set - 0x5000 indicates a TCP header length of 5x(32-bit) words, and no reserved bits set or flags. The ACK and SEQ numbers are also both zero as we would expect.

This kind of packet is crafted to evade some packet filtering firewalls which often look for inbound SYN packets, but let through other TCP packets without inspecting them. The technique relies on the IP stacks of the target hosts following the RFC and sending back a reset on a port they are not listening, hence by deduction one can find the listening ports. It can also interestingly be used against hosts that do not follow the RFC as a means of OS identification, notably windows95/NT machines.

[107] Rain Forest Puppy - www.wiretrip.org
[108] Nmap is available from www.insecure.org.

The following two source-target pairs accounted for over 4000 of the 7,393 'Null Scan' alerts.

Alert Message	Source IP	Dest IP	Total
Null scan!	64.123.89.20 5	MY.NET.252.134	1566
Null scan !	216.78.252.2 20	MY.NET.222.54	2465

Other alerts for 64.123.89.205 are:

Alert Message	Src IP	Src Port	Dst IP	Dst Port	Totals
Queso fingerprint	64.123.89.205	54666	MY.NET.252.134	31532	2
SYN-FIN scan!	64.123.89.205	60988	MY.NET.252.134	24814	3
Probable NMAP fingerprin t attempt	64.123.89.205	0	MY.NET.252.134	0	9
Null scan!	64.123.89.205	0	MY.NET.252.134	0	1566

It certainly looks as if this host is targeting MY.NET.252.134 with some reconnaissance scans.

64.123.89.205 appears to be a broadband user in the USA. (possibly Texas looking at some traceroutes)

```
# nslookup 64.123.89.205
 name = adsl-64-123-89-205.dsl.snantx.swbell.net
```

216.78.252.220 only figures in the Null scan alerts, but is also an broadband user somewhere in the USA.

```
# nslookup 216.78.252.220
 name = adsl-78-252-220.mia.bellsouth.net
```

Recommendation: Any traffic between 64.123.89.205 and MY.NET.252.134 should be investigated. If no records of traffic are kept future traffic should be monitored. Also MY.NET.252.134 should be examined for evidence of compromise.

Why is this host being targeted? If it holds classified research data etc., then it would be wise to restrict access to this host more rigorously. Eg block incoming connections from the Internet.

Alert #14 "EXPLOIT x86 NOOP"

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

or

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 NOOP"; content:"|61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61|"; classtype:shellcode-detect; sid:1394; rev:3;)

Snort SID: 648, , 1394, arachnids181

Alerts: 5,387

Unique Hosts - SRC: 190 **DST:** 161

This alert indicates a packet that includes a NOOP sled, used to pad out a

packet before some malicious code, designed to exploit a buffer overflow vulnerability.

However this signature is notoriously noisy, generating a significant number of false positives. This is particularly the case when there are large numbers of binary downloads taking place.

This alert triggered for 113 different destination ports. If we assume binary downloads from servers[109] are false positives this leaves us with the following alerts that triggered on a port < 1024.[110]

Alert Message	Dst Host	Src port	Dst Port	hits
EXPLOIT x86 NOOP	MY.NET.242.106	1112	413	2
EXPLOIT x86 NOOP	-	56464	0	2

The port zero traffic could well be a fragmentation issue (notice there is no destination host either). I really wish I could see some packets. The traffic to port 413 is listed as SMSP or storage management protocol, and I can find no buffer overflow vulnerabilities relating to this port. It looks like these alerts are all false positives.

Recommendation: The IDS should be tightened to not trigger on this alert particularly for ports which are known to generate a large number of false positives, such as FTP etc... This can be accomplished simply by not including the shellcode rules at all in the config, or tightening up the SHELLCODE_PORTS variable.

OOS Analysis

Top ten source host, destination IP pairs with OOS packets detected by the IDS.

Source IP	Destination IP	Destination Port	Total alerts
130.136.4.208	MY.NET.217.54	6346	4573
66.117.21.91	MY.NET.224.134	1182	616
209.123.49.137	MY.NET.195.155	6885	387
213.197.10.95	MY.NET.223.46	6882	359
148.63.137.221	MY.NET.235.186	1852	358
148.63.151.3	MY.NET.235.202	3516	317
210.253.206.180	MY.NET.211.26	6011	289
209.123.49.137	MY.NET.218.254	6882	282
209.123.49.137	MY.NET.226.178	6881	239

Top Ten OOS generating ports and IPs

Dst Port	*Count*		*Src IP*	
6346	5,667		130.136.4.208	4573
25	3,170		209.123.49.137	1194
1214	1,871		66.117.21.91	903
80	1,383		213.197.10.95	370
1182	1,000		148.63.137.221	368
6882	772		66.140.25.156	349
4662	641		148.63.151.3	318
6881	545		210.253.206.180	289
6011	457		213.186.35.9	244

[109] The alerts removed where for ports associated with: ftp, samba/netbios, http, directconnect, KaZaa, nntp,

[110] An exploit of this nature is highly targeted as a buffer overflow exploit against a program running usually with elevated privileges. It seems likely that the program would therefore be allocated a 'well known' port in the 1- 1023 range.

Dst Port	Count		Src IP	
1852	359		193.230.240.106	235
Total	15,865			8,843

There is very little outbound OOS traffic according to the logs, in fact only 66 instances out of nearly 20,000, almost all coming from MY.NET.12.4 .

Of the inbound OOS traffic the majority of the alerts are generated from traffic from 130.136.4.208 to MY.NET.217.54, over 4500 separate packets. In fact MY.NET.217.54 was involved in OOS packet alerts from 18 different external hosts. The packets from 130.136.4.208 came in two phases. Firstly on 06/05 between 13:09 and 13:39, then on 07/05 between 04:40 and 10:58.

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

05/06-13:09:33.325946 130.136.4.208:3083 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:46208 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xFF5BC5BC  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54358332 0 NOP WS: 0

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

05/06-13:09:33.363125 130.136.4.208:3092 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:7530 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xFEDC6C55  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54358335 0 NOP WS: 0

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

05/06-13:10:03.977813 130.136.4.208:3301 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:2164 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x1247541  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54361397 0 NOP WS: 0

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

05/06-13:10:19.238780 130.136.4.208:3391 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:6371 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x1DF9BCF  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54362924 0 NOP WS: 0

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

05/06-13:10:19.239554 130.136.4.208:3392 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:63433 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x1D874E5  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54362924 0 NOP WS: 0

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

05/06-13:10:49.759211 130.136.4.208:3586 -> MY.NET.217.54:6346
TCP TTL:53 TOS:0x0 ID:6267 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x3E29763  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 54365976 0 NOP WS: 0
```

Port 6346 is the IANA allocated port for Gnutella-svc, more peer-to-peer file sharing evidence! In the excerpts above the two high order Reserved bits have been set in the TCP header. These are the TCP ECN bits, and are set when ECN is employed to indicate congestion. In this case the '12' preceding the TCP flags indicates both bits are set which means that the ECN-chow bit is set notifying the receiver of congestion as well as the CWR bit which indicates that the sender has cut it's congestion window.[111] Was this p2p software simply congesting the network which contained ECN enabled routers and nodes?

Nearly 15,000 of the 18,500 OOS packets had one of these two bits set, the majority having both set as in the above examples. There seemed to be no real pattern to this traffic apart from that one third of it was destined for MY.NET.217.54, and took place in the two specified time slots.

SCANS

The scans file was able to help clarify many of the alerts generated in the alert logs. Looking at them in isolation for other anomalous behavior I decided to split the scans into 'ingress' and 'egress' groups.

'Egress' scans:

There was a total of 1,998,621 scans from internal hosts during this short 5 day period, the great majority of these are from one internal host scanning for hosts listening on UDP port 17300[112]. Below is a quick summary of the top 15 scans grouped by destination port (and target protocol) and ordered by frequency.

Proto	Port	Probable service	Hits	Proto	Port	Probable service	Hits
UDP	20100	Soldier of Fortune II (Game)	3,364	SYN	80	WWW / HTTP	13,882
UDP	27005	HalfLife/CounterStrike (Game)	3,627	UDP	0	Unix Port allocation [113]	14,266
UDP	6346	Gnutella-svc (file sharing)	3,683	UDP	22321	Backdoor.Dobol (Trojan)	17,057
UDP	41170	"Blubster" (file sharing)	3,797	UDP	6257	WinMX (file sharing)	28,813
UDP	14690	“Battlefield 1942” (Game)	4,426	UDP	53	Domain Name Service	35,983
UDP	4672	Remote file access server	4,595	UDP	137	Netbios Name Server	65,587
UDP	1214	KAZAA	7,845	SYN	17300	Kuang2 (Virus)	1,340,272
UDP	7674	iMQ SSL tunnel	9,625				

Most of the outgoing scans appear to be for either gaming or file sharing. The exceptions would be the scans for web servers, Dobol, DNS and Kuang2.

The file sharing scans alone encompass more than 250 different internal hosts,[114] and totals more than 45,000 scans. One quarter of these from one host 130.85.207.230 searching for WinMX[115] hosts.

Also of interest is 130.85.202.238 who scanned 213.64.169.124 for 45,530

[111] Floyd, S

[112] 'The Super scanner' in my Top Ten Talkers section

[113] In Unix network programming, specifying port 0 is used to ask for a dynamic port, ie the next freely available port. This does not work on 'MS Windows' and can therefore be used to help identify the OS of a host.

[114] This is from the top 15 scans only, and only includes outbound scans. I have not included the UDP 137 scans here as they could be an indicator of worm activity rather than simply file sharing.

[115] WinMX is a FREE file-sharing program, it utilises a peer to peer network similar in concept to Napster or KaZaa.

different ports in one 18 hour period.

Recommendation: The hosts performing the scans for DNS, Dobol, Kuang2, or Web Servers should be checked for abuse of University policy or possible compromise. It would also seem that a great deal of university bandwidth is being used by peer-to-peer file sharing utilities. This could prove very difficult to curb without imposing much stronger border filtering, such as only allowing explicitly permitted traffic in and out, and blocking everything else. Another option could be to limit the bandwidth available to single hosts to say 64k, unless there is a specific need. This however would require some form of traffic shaping at the perimeter.

Hosts actively scanning suspicious ports:[116]

Src IP	Dst port	Hits	Src IP	Dst port	Hits
130.85.168.109	22321	1,353	**130.85.97.36**	137	3,766
130.85.153.187	22321	1,419	**130.85.97.233**	137	4,423
130.85.137.7	53	2,306	**130.85.97.65**	137	5,694
130.85.97.143	80	2,528	**130.85.97.34**	80	6,955
130.85.235.110	0	2,622	**130.85.168.177**	22321	7,619
130.85.242.250	22321	4,552	**130.85.202.238**	0	14,236
130.85.97.172	137	3,323	**130.85.97.83**	17300	22,664
130.85.97.97	137	3,335	**130.85.1.3**	53	33,658
			130.85.196.193	17300	1,340,272

Ingress Scans:

There were a total of 289,099 inbound scans over the period covered by these logs. I was more concerned with the outbound scanning in general as it is an indication of suspicious behavior, bandwidth abuse or compromised hosts. Scanning from the Internet is something that is always going to occur. The amount of scans showing up though is perhaps a further indication of the inadequacy of the perimeter filtering. The majority of these scans should never make it into the University's network. As shown in the table below the scanning is diverse with the vulnerability de-jour seemingly being the MS LANMAN DOS attack as outlined in BUG-ID: 2002011[117], followed by a list of the usual suspects, file sharing applications etc..

Port	Probable service	Hits	Port	Probable service	Hits
6112	dtspcd	1,018	554	Real Time Stream Control Protocol	4,268
21	FTP control	1,502	139	NETBIOS Session Service	5,270
6346	gnutella-svc	1,581	0	Unix Port allocation	6,200
22	telnet	2,558	135	DCE endpoint resolution	11,728
6588	AnalogX Proxy	2,904	17300	Kuang2	17,120
8080	WWW / HTTP	2,934	1080	SOCKS	21,550
3128	Squid Proxy	2,996	1433	Microsoft-SQL-Server	21,990
8000	iRDMI	3,058	80	WWW / HTTP	34,140
25	SMTP	3,477	445	Microsoft-DS	130,712
4000	Skydance	3,923			

Recommendation: Many of these machines could simply be infected with a worm or virus trying to find other vulnerable hosts to infect. Scans such as these should be blocked at the University's perimeter. If the University runs

[116] Grouped by srcIP:dstPort and ordered by frequency.

[117] BUG-ID: 2002011 The default LANMAN registry settings on Windows 2000 could allow a malicious user, with access to TCP port 445 on your Windows 2000, to cause a Denial of Service.

hosts that are vulnerable to any of the attacks these scans are looking for, those machines should be examined and patched immediately.

Top Talkers

The Squeaky Wheel - MY.NET.202.238

The majority of the traffic for MY.NET.202.238 is destined to 213.64.169.124 on port 0. This was analysed in greater detail as part of my investigation of the " Incomplete Packet Fragments Discarded " alerts.

Alert message	Src port	Dst Host	Dst Port	Hits
Incomplete Packet Fragments Discarde d	0	213.64.169.124	0	316283

There is also a small amount of UDP 56464 traffic generating alerts. This could be related to the NLANR/DAST Multicast Beacon[118] project. The IP in question resolves to resnet1-133, presumably a student's dormitory PC.

This traffic all seems to be benign in nature, although extremely noisy.

So small yet so noisy - MY.NET.235.110

The scan logs are full of this address scanning for port 0, often with strange options. Looks like information gathering except that it is so noisy, definitely a stealth scan in name only[119]. The scans start in the early hours of the morning of the 7th and continue almost completely unabated for 92 hours through till late on the 11th. The packets often have reflexive ports just to make them even more peculiar, yet none are strange enough to make it into the OOS logs. They are getting picked up by snort as 'Tiny Fragments - Possible Hostile Activity' which explains the lack of TCP information.

The Super Scanner - 130.85.196.193

The host 130.85.196.193 scanned a total of 446,462 hosts in 489 different scans. There were also a large number of SMB traffic to this host, with 329 wildcard alerts triggered by this IP. All bar one are TCP scans. There are also a few 'IRC user /kill detected possible trojan' alerts for this IP. In total it is listed 1,341,176 times in the scan logs, 99% of those are for scans for port 17300.

> Port 17300 is the standard port for ".. a backdoor trojan called "Kuang2 The Virus." [120]

This is an old virus borne trojan, circa 1999. It seems most likely that this host is scanning for infected machines with the back door at port 17300 using some form of automated scanning tool.

Other interesting traffic for this host (from scan logs),

Service	KaZaa	Web	IRC
Hits	2	36	814

The port 80 traffic is no doubt web oriented, perhaps the user is looking for a

[118] NLANR/DAST Multicast Beacon - active measurement software that monitors the performance of a multicast sessions run by The National Laboratory for Applied Network Research (NLANR).

[119] Stealth scans is a general name for scans other than syn scans, the idea being you look for closed ports rather than open ports, then assume the rest are there but filtered.

[120] Patz, Kevin

web server to compromise or infect, or maybe a proxy server, but there are no scans for 1080 or 3128 other ports associated with proxy servers such as SOCKs or squid.

The 1214 traffic is most probably KaZaa file sharing traffic. While 6667 is the well known port for IRC. So it would seem 130.85.196.193 is a user of both IRC and KaZaa, or perhaps the machine is infected with a trojan that is seeking an IRC server to join to await instructions?

We do not see much in the way of traffic from any of the scanned hosts, except for a few entries in the scan logs.

```
TCP SRC and DST outside network [**] 0.0.0.0:50023  -> 64.12.185.119:80
TCP SRC and DST outside network [**] 0.0.0.0:49966  -> 64.12.185.119:6667
PORTSCAN DETECTED from 64.12.174.249 (STEALTH) [**]
PORTSCAN DETECTED from 64.12.174.185 (STEALTH) [**]
PORTSCAN DETECTED from 152.163.208.185 (STEALTH) [**]
```

One good turn deserves another, so perhaps some of the scanned hosts were curious?

Recommendation: I strongly recommend further investigation of the internal host 130.85.196.193 . It is being used in a manner that should be contrary to the University's policy[121], or has been compromised.

The File-share King - 130.85.207.230

130.85.207.230 was responsible for nearly 12,000 scans aimed at known peer-to-peer application ports, notably WinMX. This one host accounts for nearly one third of all WinMX scans during this five-day period. They also logged alerts for scans for Napster and a number of common on line games. The IP resolves to resnet1-442.resnet.umbc.edu, which indicates it is a student's dormitory PC. The usage of the five days was fairly bursty with most activity on the morning of the 7th.

WinMX protégé. 130.85.205.178

Not the champ of WinMX scans but certainly the start of a concerted effort to back up the entire Internet. This IP is responsible for nearly 5,000 scans for WinMX. This user is almost as dedicated as the NWN user below racking up 22 hours of solid searching and leeching, covering all five days,. This included an almost consistent 14-hour stretch on the 9th.

The Gamers

The following are using large quantities of university bandwidth to play on line games.

MY.NET.201.58 - resnet1-24

NeverWinter Nights is the game of choice for this gamer be it the middle of the

[121] I do not have access the University's policy pertaini ng to the use of its network and equipment so I cannot be sure.

day or late into the night. This user is using up countless amounts of the University's bandwidth feeding their role-playing addiction. No less than 58 hours during the five days! Usually logging on at about 9am and continuing on till one or two the following morning.

130.85.197.66 - does not resolve.

This is our 'Battlefield 1942' player. Another dedicated on line gamer using University bandwidth to feed their addiction. This time a more moderate 24 hours of on line gaming in a 5-day period. This user is somewhat more organised, spending four to five hours on average in the evenings playing this particular online game.

The IRC Bot - 130.85.195.99

This host does not resolve via DNS, but it is not in the 'resnet' network, which makes it suspicious that it is using IRC so much. Added to this the number of user /kill alerts it generates, this user is either a world champion at offending IRC admins or more likely an IRC bot of some kind. Either way this IP is responsible for over 17 thousand alerts in a 5-day period, almost entirely for IRC bot related activity.

Tftp user - MY.NET.240.10

MY.NET.240.10 is involved in a great deal of TFTP traffic to a number of hosts in the 64.12/16 network which is registered to AOL. In total over the five days this host generated over 5000 alerts for external TFTP access. From the early hours of 7th till the early hours of 12th, spanning the entire five day period. The IP in question is another 'rasnet' host, so likely a student's dormitory PC. I cannot tell why they are accessing these external TFTP servers so much but it should certainly be looked into.

130.85.1.3 - DNS Server?

This host generated nearly 34,000 hits in the scan logs, almost all to port 53. Is this a malicious user trying to find vulnerable BIND servers or just a busy name server?

The IP resolves to UMBC3, and it is in a suspicious subnet, suggesting that it may indeed be a hardworking campus server rather than a narcoleptic student hacker.

A quick check using dig reveals it is serving DNS to the university and indeed the internet at large:

```
$ dig @130.85.1.3 www.umbc.edu
  ;; QUESTION SECTION:
  ;www.umbc.edu.                  IN      A

  ;; ANSWER SECTION:
  www.umbc.edu.           86400   IN      A       130.85.24.34

  ;; AUTHORITY SECTION:
  umbc.edu.               86400   IN      NS      UMBC3.umbc.edu.
  umbc.edu.               86400   IN      NS      UMBC4.umbc.edu.
```

umbc.edu.	86400	IN	NS	UMBC5.umbc.edu.
;; ADDITIONAL SECTION:				
UMBC3.umbc.edu.	86400	IN	A	130.85.1.3
UMBC4.umbc.edu.	86400	IN	A	130.85.1.4
UMBC5.umbc.edu.	86400	IN	A	130.85.1.5

Thankfully my attempt to complete a zone transfer was denied, as was my check for version bind.

```
$ dig @130.85.1.3 txt chaos version.bind
version.bind.                0       CH      TXT     "Yeah
Right"
```

Five External Candidates

Adore Worm infection or Role playing addiction?

This machine was investigated as part of my analysis of the 'High port 65535' alerts.

Trying whois -h whois.arin.net **66.42.68.210**

```
OrgName:    Pac-West Telecomm, INC.
OrgID:      PWTI
Address:    1776 W. March Lane
Address:    Suite 250
City:       Stockton
StateProv:  CA
PostalCode: 95207
Country:    US
NetRange:   66.42.0.0 - 66.42.127.255
CIDR:       66.42.0.0/17
NetName:    MDSG-PACWEST-1BLK
NetHandle:  NET-66-42-0-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.MDSG-PACWEST.COM
NameServer: NS2.MDSG-PACWEST.COM
NameServer: NS3.MDSG-PACWEST.COM
NameServer: NS4.MDSG-PACWEST.COM
NameServer: NS5.MDSG-PACWEST.COM
NameServer: NS6.MDSG-PACWEST.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-11-10
Updated:    2003-05-28
TechHandle: ZP86-ARIN
TechName:   Administrator
TechPhone:  +1-800-722-9378
TechEmail:  admin@mdsg-pacwest.com

OrgTechHandle: ZP86-ARIN
OrgTechName:   Administrator
OrgTechPhone:  +1-800-722-9378
OrgTechEmail:  admin@mdsg-pacwest.com
```

DoS Fragger?[122]

213.64.169.124 **-** h124n2fls33o812.telia.com

inetnum: 213.64.0.0 - 213.64.255.255
netname: TELIANET
descr: Telia Network services
descr: ISP
country: SE
admin-c: TR889-RIPE
tech-c: TR889-RIPE
status: ASSIGNED PA
notify: mntripe@telia.net
notify: backbone@telia.net
mnt-by: TELIANET-LIR
changed: amar@telia.net 20010404
changed: aca@telia.net 20020109
source: RIPE
route: 213.64.0.0/14
descr: TELIANET-BLK
origin: AS3301
mnt-by: TELIANET-RR
changed: rr@telia.net 20010405
source: RIPE
role: TeliaNet Registry
address: Telia Network Services
address: Carrier & Networks
address: Box 10707
address: SE-121 29 Stockholm
address: Sweden
fax-no: +46 8 4568935
e-mail: ip@telia.net
e-mail: registry@telia.net
e-mail: dns@telia.net
e-mail: backbone@telia.net
admin-c: AA90-RIPE
tech-c: AA90-RIPE
tech-c: LK221-RIPE
tech-c: YL39-RIPE
tech-c: IC106-RIPE
tech-c: ACA-RIPE
tech-c: UL302-RIPE
tech-c: EC1084-RIPE
tech-c: JS7984-RIPE
tech-c: OE207-RIPE
tech-c: EER2-RIPE
tech-c: RR6890-RIPE
tech-c: PJ2540-RIPE
tech-c: SH10271-RIPE
tech-c: FIA-RIPE
tech-c: IF264-RIPE
tech-c: BM2022-RIPE
tech-c: LS483-RIPE
tech-c: AF145-RIPE
nic-hdl: TR889-RIPE
notify: mntripe@telia.net

[122] Investigated as part of Alert #1 "Incomplete Packet Fragments Discarded"

mnt-by: TELIANET-LIR
changed: fia@telia.net 20020319
changed: eva@telia.net 20020821
source: RIPE

Mr Nosey

Trying whois -h whois.arin.net 64.123.89.205 [123]

OrgName: SBC Internet Services - Southwest
OrgID: SBIS
Address: 2701 W 15th St PMB 236
City: Plano
StateProv: TX
PostalCode: 75075
Country: US
NetRange: 64.123.0.0 - 64.123.255.255
CIDR: 64.123.0.0/16
NetName: SBIS-4BL
NetHandle: NET-64-123-0-0-1
Parent: NET-64-0-0-0-0
NetType: Direct Allocation
NameServer: NS1.SWBELL.NET
NameServer: NS2.SWBELL.NET
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
Comment: please send all abuse issue e-mails to abuse@swbell.net
RegDate: 2000-07-10
Updated: 2000-07-10
TechHandle: ZS44-ARIN
TechName: IPAdmin-SBIS
TechPhone: +1-888-212-5411
TechEmail: IPAdmin-SBIS@sbcis.sbc.com
OrgAbuseHandle: ABUSE6-ARIN
OrgAbuseName: Abuse - Southwestern Bell Internet
OrgAbusePhone: +1-877-722-3755
OrgAbuseEmail: abuse@swbell.net
OrgNOCHandle: SUPPO-ARIN
OrgNOCName: Support - Southwestern Bell Internet Services
OrgNOCPhone: +1-888-212-5411
OrgNOCEmail: support@swbell.net
OrgTechHandle: IPADM2-ARIN
OrgTechName: IPAdmin-SBIS
OrgTechPhone: +1-888-212-5411
OrgTechEmail: IPAdmin-SBIS@sbis.sbc.com

How nice the SBC Internet Services people have included an abuse mail address in their whois database entry.

Introducing Mr and Mrs Nimda

The following two hosts are suspected of infecting University hosts with Nimda.

80.17.44.90:

inetnum: 80.17.44.88 - 80.17.44.95
netname: COMUNEDIFIUGGI

[123] Investigated as part of Alert #13 "Null scan!"

```
descr:       COMUNEDIFIUGGI
country:     IT
admin-c:     MT1677-RIPE
tech-c:      MT1677-RIPE
status:      ASSIGNED PA
mnt-by:      INTERB-MNT
notify:      network@cgi.interbusiness.it
changed:     network@cgi.interbusiness.it 20030402
source:      RIPE
route:       80.16.0.0/15
descr:       INTERBUSINESS
origin:      AS3269
remarks:     Send report of network abuse/spam
remarks:     only to: abuse@interbusiness.it .
remarks:     If you report abuse to any other address
remarks:     you will get no response.
notify:      network@cgi.interbusiness.it
mnt-by:      INTERB-MNT
changed:     mattu@cgi.interbusiness.it 20011009
source:      RIPE
person:      MARCO TURRIZIANI
address:     COMUNE DI FIUGGI
address:     PIAZZA TRENTO E TRIESTE 1
address:     MARCO TURRIZIANI
address:     Italy
phone:       +39 077554611
fax-no:      +39 077554611
nic-hdl:     MT1677-RIPE
changed:     domain@cgi.interbusiness.it 20030402
source:      RIPE
```

Another Registration with an abuse email address listed.

61.171.133.120

```
inetnum:     61.169.0.0 - 61.171.255.255
netname:     CHINANET-SH
descr:       CHINANET Shanghai province network
descr:       Data Communication Division
descr:       China Telecom
country:     CN
admin-c:     CH93-AP
tech-c:      XI5-AP
mnt-by:      MAINT-CHINANET
mnt-lower:   MAINT-CHINANET-SH
changed:     hostmaster@ns.chinanet.cn.net 20001201
status:      ALLOCATED PORTABLE
source:      APNIC
person:      Chinanet Hostmaster
address:     No.31 ,jingrong street,beijing
address:     100032
country:     CN
phone:       +86-10-66027112
fax-no:      +86-10-66027334
e-mail:      hostmaster@ns.chinanet.cn.net
e-mail:      anti-spam@ns.chinanet.cn.net
nic-hdl:     CH93-AP
mnt-by:      MAINT-CHINANET
changed:     hostmaster@ns.chinanet.cn.net 20021016
```

```
source:     APNIC
person:     Wu Xiao Li
address:    Room 805,61 North Si Chuan Road,Shanghai,200085,P RC
country:    CN
phone:      +86-21-63630562
fax-no:     +86-21-63630566
e-mail:     ip-admin@mail.online.sh.cn
nic-hdl:    XI5-AP
mnt-by:     MAINT-CHINANET-SH
changed:    ip-admin@mail.online.sh.cn 20010510
source:     APNIC
```

Analysis process

I used two different approaches for the analysis of the data files provided by the University.

1. For the alert, and scan files I used a number of Perl scripts[124] along with standard Unix commands such as awk, sed, uniq, grep and sort etc... to isolate and identify data of interest.
2. For the OOS files I decided to import the data into a database[125] and use SQL queries to extract the information. This technique was chosen in order to compare this technique with the earlier methods employed with the alert files.

Later on I also imported the alert files into the database as this allowed for quick generation of 'top 10' like tables for patterns of interest.

With both techniques I first concatenated the data files into a single file for each type. I then parsed the files to create CSV files with a standard layout for each source file type using some custom Perl scripts. Getting the OOS files into a standard one line CSV format proved to be less than trivial with it's pseudo random format[126] in the log files.

The differing techniques both had their advantages and drawbacks. The Perl scripts I found to be more flexible, in that I could extract exactly what I wanted. This is a reflection of the fact that I prefer Perl to SQL, and also that the scripts were custom to my needs. The database solution however had serious performance advantages. Where as the scripts could take up to 2 or 3 minutes to run, a query could be run in few tens of a second. If one is dealing with large amounts of data, that is weeks or months rather than a few days then the database approach would be invaluable.

Alert Files:

These were probably the most interesting, so I tackled these first. Once I had my CSV file I had to decide what information I wanted to extract. I decided that grouping information on attack signature would be useful, as would grouping alerts based on source IP. The information would need to include:

[124] See the Appendixes for examples of some of the scripts used.
[125] MySQL - http://www.mysql.com
[126] Entries are on more than one line, some have a TCP, some are fragments, some have a data segment others do not etc... Makes for a pretty regexp.

- Totals for each type of attack.
- Source hosts per attack.
- Target hosts per attack.
- Target ports per attack.
- Totals for attacks from a particular host.
- Destinations for 'Attack:SrcHost' tuples.

The files contained a number of entries, which appeared to be corrupted such as the following:

```
:33259 -> 233.2.171.105/07-16:22:01.118894  [**] SMB Name Wildcard
[**] 64.170.51.119:56464
```

Where possible these alerts were tidied up by pre-parsing, however some were beyond repair and were ignored in the analysis, as it was not possible to tell for certain how the data was corrupted. Out of 1,242,686 lines of data 248, 757 were corrupted beyond repair.

OOS Files:

The OOS files were analysed by first parsing into CSV format then importing into a MySQL database. They were mainly used to corroborate the deductions of the alert files.

Scans Files:

These were also imported into the database as with the OOS data. This data was used to identify p2p users and other network hogs, as well as corroborate some of the conclusions of the alerts files such as nimda/code red infections. It also gave an insight into indication of prior reconnaissance however often this is done well in advance of the attack so older scan logs would have been required to make better use of this type of evidence. Given the size of 5 days of logs this could become a logistical issue.

Corroboration:

Looking at the files in isolation gave up a wealth of information, however piecing together a clear pattern of behavior often required relating data from different sources.

One of the first things I was keen to identify was the host network. In all the alert files there was a great deal of traffic to and from hosts within MY.NET.0.0/16, which seemed to be the obfuscated protected network address. To be sure I looked at a number of custom alerts such as:

> Notify Brian B. 3.56 tcp, External RPC call, TFTP - Internal UDP connection to external tftp server, MY.NET.30.3 activity, connect to 515 from outside, TCP SRC and DST outside network, TFTP - Internal TCP connection to external tftp server, [UMBC NIDS IRC Alert] etc...

These all pointed to MY.NET.0.0/16 as being the home network. To confirm this I looked at the OOS packets, as these are the only whole packets I had. There are a number of requests to web servers containing the string http://www.umbc.edu. Now this resolves to 130.85.24.34, so is MY.NET.0.0/16 really 130.85.0.0/16? To discover this I needed to check the scan logs. I had

noticed a high occurrence of 'Null scan' alerts from 64.123.89.205 to MY.NET.252.134, there should be evidence of this in the scan logs which I had noticed did not contain any reference to MY.NET. A quick grep of the scans file for alerts from 64.123.89.205 generated the following:

```
# grep "64.123.89.205:0 ->" ../scans-combined  | grep NULL | awk '{print $6}' | awk -F:
'{print $1}' | sort -n | uniq
130.85.252.134
```

Bingo! The only 'Null scan' alerts from 64.123.89.205 in the scan logs are to 130.85.252.134, which must correlate to MY.NET.252.134 so we can confirm the private network MY.NET is indeed 130.85.0.0.

Trying whois -h whois.arin.net 130.85.0.0

OrgName: University of Maryland Baltimore County
OrgID: UMBC
Address: UMBC University Computing
City: Baltimore
StateProv: MD
PostalCode: 21250
Country: US
NetRange: 130.85.0.0 - 130.85.255.255
CIDR: 130.85.0.0/16
NetName: UMBCNET
NetHandle: NET-130-85-0-0-1
Parent: NET-130-0-0-0-0
NetType: Direct Assignment
NameServer: UMBC5.UMBC.EDU
NameServer: UMBC4.UMBC.EDU
NameServer: UMBC3.UMBC.EDU
Comment:
RegDate: 1988-07-05
Updated: 2000-03-17
TechHandle: JJS41-ARIN
TechName: Suess, John J.
TechPhone: +1-410-455-2582
TechEmail: jack@umbc.edu

References[i]

1.Bontrager , William - JavaScript Tutorial Part II URL: http://www.web-source.net/javascript_tutorial2.htm (July 05 2003)

2.Carpenter, Carlin - "GCIA Practical." Mar 24 2002
URL:http://www.giac.org/practical/Carlin_Carpenter_GCIA.doc (June 28 2003)

3.CERT® Advisory CA-2002-18 OpenSSH Vulnerabilities in Challenge Response Handling - December 6, 2002. URL: http://www.cert.org/advisories/CA-2002-18.html (May 05 2003)

4. Computer Networking, Port 0 – TCP UDP Port Number Glossary
URL:http://compnetworking.about.com/library/ports/blports_0.htm (July 1 2003)

5.Dell, J Anthony. "Adore Worm – Another Mutation" Apr, 6 1001.
URL:http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf (June 26 2003)

6. EA Games, 'BattleField1942' -
URL:http://www.eagames.com/official/battlefield1942/editorial/community_message_26.jsp

[i]Combined references including references for section 1.

(July 1 2003)

7.'Ed3f' - Firewall spotting and networks analysis with a broken CRC', Phrack Volume60 URL: http://www.phrack.org/show.php?p=60&a=12 (5 April 2003)

8. Floyd, Sally - TCP and Explicit Congestion Notification URL:http://www.icir.org/floyd/papers/tcp_ecn.4.pdf (July 08 2003)

9.Fyodor - "Idle Scanning and related IPID games" – Insecure.org. URL: http://www.insecure.org/ (May 05 2003)

10.GameRanger, Help – 'Ports for Hosting' URL: http://www.gameranger.com/help/ports/ (July 1 2003)

11.Goldsmith, David and Schiffman, Michael - Firewalking A Traceroute-Like Analysis of IP Packet Responses to Determine Gateway Access Control Lists. URL: http:www.packetfactory.net/firewalk/firewalk-final.pdf (9 April 2003)

12.Gordon, Les M - "GCIA Practical." Nov 22 2002 URL:http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc (June 28 2003)

13.Granier, Thomas B - LOGS: GIAC GCIA Version 3.3 Practical Detect - Reserved bit set, but why? URL: http://cert.uni-stuttgart.de/archive/intrusions/2002/11/msg00256.html (July 11 2003)

14. Hickey, Shane - [Snort-users] WEB-CLIENT javascript URL host spoofing attempt URL:http://msgs.securepoint.com/cgi-bin/get/snort-0211/427.html (July 7 2003)

15. Hobbit – Netcat 'Hobbit Documentation' URL:http://www.zoran.net/wm_resources/netcat_hobbit.asp (July 05 2003)

16.IANA well known ports listing. URL: http://www.iana.oeg/assignments/port-numbers (May 05 2003)

17.Incident Report - Mynetwatcman. URL: http://www.mynetwatchman.com/mynetwatchman/ListIncidentsbyIP.asp (May 07 2003)

18. Internet Storm Centre – URL: http://isc.incidents.org/port_details.html?port=137 (26 June 2003)

19. Kalt, C – RFC2812 'Internet Relay Chat: Client Protocol' April 2000 URL:http://www.kvirc.de/docu/doc_rfc2812.html (July 1 2003)

20."MAC address" - Webopedia. URL: http://www.webopedia.com/TERM/M/MAC_address.html (May 09 2003)

21. McDonald, Tim - Adore'-able Worm Targets Linux URL:http://www.newsfactor.com/perl/story/8738.html (June 26 2003)

22. MyNetWatchman –Incident Report. URL:http://www.mynetwatchman.com/LID.asp?IID=30378477 (June 27 2003)

23. Nazario, Jose - Security Incidents: Re: ICMP Source Quench - Can it be some flood attack? URL:http://lists.insecure.org/incidents/2000/Sep/0060.html (July 10 2003)

24. Nelson, Nick - URL: http://arpa.com/~nick/snort (June 22 2003)

25. NetAPP - NetCache Hardware Product Family URL:http://www.netapp.com/products/netcache/netcache_family.html (July 09 2003)

26.Northcutt, Stephen & Novak, Judy - Network Intrusion Detection, an Analyst's Handbook - Second Edition - ??? - 19??

27. Patz, Kevin. "Re: Port 17300 probes?" - Incidents mailing list <securityfocus.com> - URL: http://cert.uni-stuttgart.de/archive/incidents/2003/04/msg00069.html (June 27 2003)

28.Postel, Jon (editor) - RFC791 - Internet protocol, DARPA internet program protocol specification0. URL: http://www.faqs.org/rfcs/rfc791.html (May 05 2003)

29. Ptacek, Thomas H. and Newsham, Timothy, "Insertion, Evasion, and Denial of Service: El uding Network Intrusion Detection", Secure Networks, Januar y 1998 URL: http://www.insecure.org/stf/secnet_ids/secnet_ids.html (20 March 2003)

30. Rain Forrest Puppy. Phrack Magazine Issue 55 – 'Perl CGI Problems' URL:http://www.phrack.org/show.php?p=55&a=7 (June 29 2003)

31. Roesch, Martin & Green, Chris - Writing Snort Rules Chapter 2 (Snort 2.0.0) URL:http://www.snort.org/docs/writing_rules/chap2. html#tth_chAp2 (July 06 2003)

32. Roesch, Marty – Snort Users Mailing list. URL: http://www.mcabee.org/lists/snort-users/Nov-01/msg00820.html (26 June 2003)

33. Sam Spade.org – Online Network tools. UR L: http://www.samspade.org/ (June 08 2003)

34. Sandblad, Andreas - 'Mozilla cookie stealin g' - Sandblad advisory #9 URL:http://www.securityfocus.com/archive/1/284012 (July 05 2003)

35. SANS Institute -Adore Worm Version 0.8 - April 12, 2001 URL:http://www.sans.org/y2k/adore.htm (June 26 2003)

36. SANS Top Vulnerabilities. URL: http://www.sans.org/top20/#U6 (May 05 2003)

37. Schmelzel, Paul – "Nimda Surviving the Hydra." URL:http://www.giac.org/practical/GCIH/Paul_Schmelzel_GCIH.pdf (June 27 2003)

38. SecurityFocus, 'Mozilla JavaScript URL Host Spoofing Arbitrary Cookie Access Vulnerability', URL: http://www.securityfocus.com/bid/5293 (July 05 2003)

39. Snort Signature Database – "SCAN NULL". URL: http://www.snort.org/snort-db/sid.html?sid=623 (June 21 2003)

40. Snort Signature Database – "SHELLCODE-X86-NOOP". URL: http://www.snort.org/snort-db/sid.html?sid=648 (June 21 2003)

41. Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994.

42. Symantic, Security Response - W32.Nimda.A@mm URL:http://www.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html (June 29 2003)

43. Upatising, Viriya – GCIA Practical URL: http://www.giac.org/practical/Viriya_Upatising.doc (July 10 2003)

44. Venema , Wietse - TCPWrappers blurb URL: ftp://ftp.porcupine.org/pub/security/index.html (June 26 2003)

45. Vision, Max - IDS181 "SHELLCODE-X86-NOOP". URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids181&view=event (June 21 2003)

46. Vision, Max - IDS4 "PROBE-NULL_SCAN". URL: http://www.snort.org/info/ids4 (June 21 2003)

47. Zirkle, Laurie - Tiny Fragments- URL:http://archives.neohapsis.com/archives/snort/2000-05/0009.html (June 26 2003)

Appendix A

Incidents.org Raw logs README

"The logs within this directory are provided for your use while completing the GCIA practical.

The log files are the result of a Snort instance running in binary logging mode. This means that only the packets that violate the rule set will appear in the log. The logs themselves have been sanitized. All of the IP addresses of the protected network space have been 'm unged'. Additionally, the checksums have been modified to prevent clever people from discovering the original IP addresses. You will find that certain keywords within the packets have been replaced with 'X's. All ICMP, DNS, SMTP and Web traffic has also been remove d. A common question is, "Are the addresses changed in the same way across all of the files?" The answer is both yes and no. If you look at the time stamp associated with the files on the website, you will see that groups of files have been posted on the same day. File s posted on the same day will have the IP addresses of the protected network modified consistently. IP addresses belonging to non-local hosts are the actual IP addresses and will be consistent across all log files regardless of date."

Appendix B

CVE ID	Proto	Source Port	Target port	Description
CVE-2001-1162	udp	any	137	Directory traversal vulnerability in the %m macro in the smb.conf configuration file in Samba before 2.2.0a allows remote attackers to overwrite certain files via a .. in a NETBIOS name, which is used as the name for a .log file.
CVE-2000-0673	tcp	any	137	The NetBIOS Name Server (NBNS) protocol does not perform authentication, which allows remote attackers to cause a denial of service by sending a spoofed Name Conflict or Name Release datagram, aka the "NetBIOS Name Server Protocol Spoofing" vulnerability.
CVE-2000-0347	udp	any	137	Windows 95 and Windows 98 allow a remote attacker to cause a denial of service via a NetBIOS session request packet with a NULL source name.
CVE-1999-0810	tcp	any	137	Denial of service in Samba NETBIOS name service daemon (nmbd).
CVE-2000-0673	udp	any	137	The NetBIOS Name Server (NBNS) protocol does not perform authentication, which allows remote attackers to cause a denial of service by sending a spoofed Name Conflict or Name Release datagram, aka the "NetBIOS Name Server Protocol Spoofing" vulnerability.
CVE-1999-0810	udp	any	137	Denial of service in Samba NETBIOS name service daemon (nmbd).
CVE-1999-0288	tcp	any	137	Denial of service in WINS with malformed data to port 137 (NETBIOS Name Service).
CVE-1999-0288	udp	any	137	Denial of service in WINS with malformed data to port 137 (NETBIOS Name Service).

Appendix C

Some of my Scripts:

```
#!/usr/bin/perl -w
################################################
# Name:      parse-alerts-simple.p
#
# Synopsis:  parse-alerts-simple.pl [-d ] [-p] -f <alert-file>
#
# Descrition:
#       Simple script to generate a csv file from
#       a snort alert file which I may later import
#       into a DB.  Also having one line of data
#       makes mining it in correlation with
#       the other files easier
#       Very similar to parse-alerts.pl, however this script only does the csv stuff and
#       Analysis is left to alert-reports.pl.
#
# Author: James Maher <scouser@paradise.net.nz>
#
################################################

use Getopt::Long;
# get the options
my $result = GetOptions ("file=s"   => \$alert_file,     # string
                         "ports"    => \$ports,          # include portscan data in csv
                         "d"        => \$debug);         # debug flag
```

```
die "usage: $0 [-d ] <alert-file>\n" if (! defined($alert_file));

# use a temp output csv file if we are in debug mode
$csv_file = $debug ? 'output.csv' : defined($ports) ? "$alert_file+scans.csv" : "$alert_file.csv";

open DATA, "$alert_file" or die "Failed to open $alert_file: $?\n";
open CSV, ">$csv_file" or die "Failed to open CSV file: $?\n";

# Data structure for storing results
my %alert_list;
while (<DATA>)
{
   if ((! /^\d/) | ( /.*\[\*{2}\].*\[\*{2}\].*\[\*{2}\]/)) {
      push (@matchless, $_) if (! /^\d/) ;
     next;
   }
   chomp $_;
   $count ++;
   # better make sure there are no commas in the data
   $_ =~ s/,//g;

   my  ($date_s,$desc,$src,$src_prt,$dst,$dst_prt) ;

   # some patterns to make the regexps easier to follow
   my $IP_CHRS = '(?:[a-zA-Z0-9\.]+)';
   my $DLM = '\[\*{2}\]';
   my $EOS = 'End of portscan from';
   my $TM = 'TOTAL time\S';

    if ($_ =~ /^(\S+)\s+\[\*{2}\]\s+(.*)\s+\[\*{2}\]\s+($IP_CHRS)(:(\d+))?(?:\s+->\s+($IP_CHRS)(:(\d+))?)?$/) {
                        ($date_s,$desc,$src)  = ($1,$2,$3);
                        if (defined($8)) {
                           ($src_prt,$dst,$dst_prt) = ($5,$6,$8);
                            print CSV "$date_s,$desc,$src,$src_prt,$dst,$dst_ prt\n";
                        }
                        elsif ( defined($6) )
                        {
                           ($src_prt,$dst,$dst_prt) = ('-', $6, '-');
                            print CSV "$date_s,$desc,$ src,$src_prt,$dst,$dst_prt\n";
                        }elsif ( defined($5) )
                        {
                           ($src_prt,$dst,$dst_prt) = ($5, '-', '-');
                           print CSV "$date_s,$desc,$src,$src_prt,$dst,$dst_prt \n";
                        }
   }
   elsif ($_ =~ /^(\S+)\s+$DLM\s+(\S+): $EOS\s+($IP_CHRS\.\d+):\s+$TM(\d+s)\S\s+hosts\S(\d+)\S TCP\S(\d+)\SUDP\S(\d+)\S\s+$DLM/) {
                        ($date_s,$desc,$src,$host_cnt)  = ($1,$2,$3,$5);
                        $port_cnt = $6 + $7;
                        print CSV "$date_s,$desc,$src,$host_cnt,$port_cnt\n" if $ports;
   }
   elsif  ($_ !~ /^(\S+)\s+$DLM\s+spp_port/ ){
                        push (@matchless, $_);
                        print STDERR "\nNo Match:\t$_\n" ;
                        next;
   }
}
close DATA or die "Failed to close data file!\n";
print "\n\t processed $count records\n\n";


#################################################################################################################
#!/usr/bin/perl -w

#################################################
# Name:      alert-reports.p
#
# Synopsis:  alert-reports.pl [-d ] [-s] [-n] [-c <max>] [-l <min>] -f <alert-file>
#
# Descrition:
#       Read in the pre parsed alert.csv file
#       and generate a number of reports
```

```
#
# Author: James Maher <scouser@paradise.net.nz>
# $Id: alert-reporter.pl,v 1.4 2003/06/22 11:49:09 scouser Exp scouser $
##################################################

use Getopt::Long;

# get the options
my $result = GetOptions ("file=s"      => \$alert_file,     # Name of the alert CSV file we are reading in
                         "portscans"   => \$scans,          # should we include portscans ?
                         "ip-order"    => \$ip_order,       # print IP summary ordered by IP not frequency
                         "summary"     => \$summary,        # Sumary mode only prints out totals of hits per IP,
                                                            # not a line for each attack type per host.
                         "verbose"     => \$verbose,        # always print out full lists rather than totals
                         "lowest=i"    => \$floor,          # what is the cut off for printing out data
                         "ceiling=i"   => \$ceiling,        # How many entires should I print out per report (ie top ten)
                         "no_attack"   => \$no_attack,      # Do not generate an attack report
                         "target"      => \$dst_grp,        # Group by target address rather than source address
                         "help"        => \$help,           # print out comand line options and die.
                         "debug"       => \$debug);         # debug flag

die "\t--ceiling <max>  How many entires should I print out per report (ie top ten)\n",
   "\t--debug            debug flag\n",
   "\t--file <name>     Name of the alert CSV file we are reading in\n",
   "\t--help             print out comand line options and die.\n",
   "\t--ip-order         print IP summary ordered by IP not frequency\n",
   "\t--lowest <min>    what is the cut off for printing out data\n",
   "\t--no_attack       Do not generate an attack report\n",
   "\t--portscans       Include portscans -- Un-implmented \n",
   "\t--summary         Sumary mode only prints out totals of hits per IP, \n",
   "\t                  not a line for each attack type per host.\n",
   "\t--target          Group by target address rather than source address\n",
   "\t--verbose         always print out full lists rather than totals\n",
   "\t--help            print out comand line options and die.\n",
   if $help;

die "usage: $0 [-d] [-s] [-n] [-c <max>] [-l <min>] -f <alert-file>\n" if (! defined($alert_file));
die "Alert file not a .csv file, aborting...\n" if $alert_file !~ /csv$/;

# Set things up before we start.
# Data structure for storing results
my %alert_list;
my $rpt_dir='/home/scouser/work/SANS/Practical/Part3/data/parsed/reports';
&initialise();

&read_in_data();

&gen_ip_report();

&gen_attack_report() if ! $no_attack;

print "\nFinished analysis, reports written to:\n$address_report\n";
print "$atck_rpt\n" if ! $no_attack;
print "\n";

##############################################################################
# Subs
##############################################################################

sub gen_ip_report() {
# Lets see what we have got ;-)
# and print out to a file
   print "\nGenerating IP report... - ";
   open TARGET, ">$address_report" or die "Could not open IP address report file\n";
   print TARGET $rpt_heading;
   $cnt =0;
   foreach $target (sort ip_sort keys (%{$alert_list{'hosts'}}))
   {
                        $cnt ++
                        &progress(200);
                        local *STDOUT = *TARGET;
```

```
                next if $target =~ /^-/;
                last if (($floor) && ($alert_list{'hosts'}->{$target}->{'count'} < $floor));
                last if (($ceiling) && ($cnt >= $ceiling));
                if ($summary){
                   print "$target\t    ".$alert_list{'hosts'}->{$target}->{'count'}."\t\t" ;

                   my @alerts = keys(%{$alert_list{'hosts'}->{$target}});
                   my $num_alrts = $#alerts;
                   print "   $num_alrts\t\t" ;
                   my @hosts = (keys (%{$alert_list{'hosts'}->{$target}->{'hosts'}}));
                   my $count = ($#hosts + 1);
                   print "  $count\t\t" ;
                   my @ports = (keys (%{$alert_list{'hosts'}->{$target}->{'dprts'}}));
                   $count = ($#ports + 1);
                   print "  $count\n" ;

                }
                else
                {
                   foreach $alert_desc (sort  keys(%{$alert_list{'hosts'}->{$target}}) )
                   {
                                #save count till after -)
                                next if $alert_desc =~ /^(count)|(hosts)|(sprts)|(dprts)/;

                                my $alrt =  $alert_list{'hosts'}->{$target}->{$alert_desc};
                                print "$target, $alert_desc, ";
                                print $alrt->{"count"}.", ";

                                my @hosts = (keys (%{$alrt->{'hosts'}}));
                                my $num_hosts = $#hosts+1;
                                if ( ($num_hosts < 3) || ($verbose)) {
                                   print "H:  ";
                                   my $last = pop @hosts;
                                   print "$_, " foreach (@hosts);
                                   print "$last ";
                                } else {
                                   print $num_hosts;
                                }
                                print "\t";

                                my @ports = (keys (%{$alrt->{"ports"}}));
                                my $num_ports = $#ports+1;
                                if ($num_ports == 0) {
                                   print "\n";
                                } elsif (($num_ports < 2) || ($verbose)) {
                                   print "P: ";
                                   my $last = pop @ports;
                                   print "$_, " foreach @ports;
                                   print "$last\n";
                                }
                                else {
                                   print "$num_ports\n";
                                }
                   }
                   print "\t".$alert_list{'hosts'}->{$target}->{'count'}." total hits ($target)\n";
                }
   }
   close TARGET || die "Could not close target tally output file\n";
}

#-----------------------------------------------
sub gen_attack_report(){
# now for the report by attack type
# pretty similar to above
   print "\nGenerating attack report... - ";
   open ATTACK, ">$atck_rpt" or die "Could not open target tally output file\n";
   print ATTACK $atck_heading;
   $cnt =0;
   foreach $attack (sort atck_sort keys (%{$alert_list{'Attacks'}} ) )
   {
                $cnt ++;
```

```
                    last if (($ceiling) && ($cnt >= $ceiling));
                    &progress(5);
                    local *STDOUT = *ATTACK ;
                    last if (($floor) && ($alert_list{'Attacks'}->{$attack}->{'count'} < $floor));
                    my $alrt =  $alert_list{'Attacks'}->{$attack};
                    print  "$attack\t".$alrt->{'count'}."\t";

                    foreach $hst ('src','dst') {
                       my @hosts = (keys (%{$alrt->{$hst}}));
                       my $num_hosts = $#hosts +1;

                       if (($num_hosts < 3) || ($verbose)) {
                                    print "H:  ";
                                    my $last = pop @hosts;
                                    print "$_, " foreach (@hosts);
                                    print "$last ";
                       } else {
                                    print "$num_hosts hosts";
                       }
                       print "\t";
                    }

                    my @ports = (keys (%{$alrt->{"ports"}}));
                    my $num_ports = $#ports+1;
                    if ($num_ports == 0) {
                       print "\n";
                    } elsif (($num_ports < 3) || ($verbose)) {
                       print "P: ";
                       my $last = pop @ports;
                       print "$_, " foreach @ports;
                       print "$last\n";
                    } else {
                       print "$num_ports ports\n";
                    }
   }
   close ATTACK || die "\nCould not close target tally output file\n";
}

#-------------------------------------------------
# sort attacks by frequency
sub atck_sort() {
   return ($alert_list{'Attacks'}->{$b}->{'count'} <=> $alert_list{'Attacks'}->{$a}->{'count'});
}

#-------------------------------------------------
# sort IP addresses
sub ip_sort {
   if (! $ip_order){
                    return -1 if (! defined($alert_list{'hosts'}->{$b}->{'count'})) || (! defined($alert_list{'hosts'}->{$a}->{'count'}));
                    return ($alert_list{'hosts'}->{$b}->{'count'} <=> $alert_list{'hosts'}->{$a}->{'count'});
   }

   # else we have to do sort by IP address ;-(
   my ($a1, $a2, $a3, $a4) = split (/\./, $a);
   my ($b1, $b2, $b3, $b4) = split (/\./, $b);

   if (($a1 =~ /^MY/) || ($b1 =~ /^MY/) ){
                    if (($a1 =~ /^MY/) && ($b1 =~ /^MY/) ) {
                       if ($a3 ne $b3) { return ($a3 <=> $b3); }
                       if ($a4 ne $b4) { return ($a4 <=> $b4); }
                    }
                    return -1 if ($a1 =~ /^MY/);
                    return 1;
   }

   if ($a1 ne $b1) { return ($a1 <=> $b1); }
   if ($a2 ne $b2) { return ($a2 <=> $b2); }
   if ($a3 ne $b3) { return ($a3 <=> $b3); }
   if ($a4 ne $b4) { return ($a4 <=> $b4); }

   return ($a <=> $b);
```

```
}

#-----------------------------------------------------
# Initialise variables based on command line args etc..
sub initialise() {
   die "WARNING: Incompatable options \'--scans\' and \'--target\' - \tCan't group portscan alerts by destination port, exiting.\n" if
(($dst_grp) && ($scans));

   print "--portscans flag not implemented, ignoring...\n" if $scans;
   undef $scans;

   $line_cnt = 0;
   print "\ninitialising report variables... \n";
   $ad_sfx = defined($summary) ? 'summary' : defined($verbose) ? 'verbose' : 'normal';
   $ad_sfx .= "-$floor" if defined($floor);
   $ad_sfx .= "-$ceiling" if $ceiling;
   $address_report = defined($dst_grp) ? "$rpt_dir/dst_IP_report.$ad_sfx" : "$rpt_dir/src_IP_report.$ad_sfx";

   $at_sfx =  defined($verbose) ? 'verbose' : 'normal';
   $at_sfx .= "-$floor" if defined($floor);
   $at_sfx .= "-$ceiling" if $ceiling;
   $atck_rpt=  "$rpt_dir/attack_report.$at_sfx";;

   $rpt_heading = "Hits per Destination Address\n=========================\n" if $dst_grp;
   $rpt_heading = "Hits per Source Address\n=======================\n" if ! $dst_grp;
   $rpt_heading .= "(including portscans)\n" if $scans;
   $rpt_heading .= "Verbose Mode\n" if $verbose;
   $rpt_heading .= "\n Address \tTot-Hits \tDistinct \tHosts \t Ports\n" if $summary;

   $atck_heading = "Attack/Scan Types\n=================\n";
   $atck_heading .= "(including portscans)\n\n" if $scans;
   $atck_heading .= "Verbose Mode\n" if $verbose;
   $atck_heading .= "\nDescription\t\t\tHits\tSrc IPs\t\tDst IPs\t\tDst Ports\n";

   print "Ceiling set to $ceiling\n" if $ceiling;
   # just for fun, and to stop you going crazy while the reports run ;-)
   @display =('-','\\','|','/');
}

#-----------------------------------------------
#
sub read_in_data() {
   print STDERR "reading in csv file ... - ";
   open DATA, "$alert_file" or die "Failed to open $alert_file: $?\n";
   while (<DATA>)  {
         &progress(1000);
         chomp;

           # no need to analyse if it is an spp portscan line unless forced to by command line flag
           my @data = split /,/;
           if ($data[1] !~ /^spp_portscan/){
                    &process_alert(@data);
                   }
                   elsif ($scans){
                     #&process_scan(@data);   # Not implemented - Handling portscans seperately now.
                   }
                   else { print "DBG: $_\n" if $debug; }
       }
   close DATA or die "Failed to close data file!\n";
}

sub process_alert() {
   my ($date_s,$desc,$src,$src_prt,$dst,$dst_prt) = @_;
   # decide who to log this under Will use the source address for now.
   if ( ! defined($dst) && ($dst_grp)){
                   print STDERR "Target grouping specified but no dst IP!! Ignoring this alert.\n" if $debug;
                   next;
   }
   my $alert_host = $dst_grp ? $dst : $src ;
   my $scnd_host  = $dst_grp ? $src : defined($dst) ? $dst : '-';
```

```
    $alert_list{'hosts'}->{$alert_host}->{$desc}={} if !$alert_list{'hosts'}->{$alert_host}->{$desc};
    my $alert = $alert_list{'hosts'}->{$alert_host}->{$desc};

    $alert->{'ports'}->{$dst_prt} ++ if ( defined($dst_prt) && ($dst_prt !~ /-/));
    $alert->{'hosts'}->{$scnd_host} ++ if $scnd_host !~ /^-/ ;
    #print $alert->{'hosts'}->{$scnd_host}."\t"  if $scnd_host !~ /^-/ ;
    $alert->{'count'} ++;

    # lets also count total alerts against a host
    $alert_list{'hosts'}->{$alert_host}->{'count'} ++;
    # and the total number of ports
    $alert_list{'hosts'}->{$alert_host}->{'sprts'}->{$src_prt} ++ if ( defined($src_prt) && ($src_prt !~ /-/));
    $alert_list{'hosts'}->{$alert_host}->{'dprts'}->{$dst_prt} ++ if ( defined($dst_prt) && ($dst_prt !~ /-/));
    $alert_list{'hosts'}->{$alert_host}->{'hosts'}->{$scnd_host} ++ if ( defined($scnd_host) && ($scnd_host !~ /-/));

    # lets also collate totals for different attacks
    $alert_list{'Attacks'}->{$desc}={} if !$alert_list{'Attacks'}->{$desc};
    local *signature = $alert_list{'Attacks'}->{$desc};
    $signature{'count'} ++;
    $signature{'src'}->{$alert_host} ++;
    $signature{'dst'}->{$scnd_host} ++ if $scnd_host !~ /^-/ ;
    $signature{'ports'}->{$dst_prt} ++ if ( defined($dst_prt) && ($dst_prt !~ /-/));
}

sub progress(){
    $mod = shift;
    if ($line_cnt % $mod == 0) {
                        $char = shift @display;
                        system("echo -n");
                        print "\b\b$char ";
                        push @display,$char;
    }
    $line_cnt ++;
}
```