



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at <http://www.giac.org/registration/gcia>



GIAC CERTIFIED INTRUSION ANALYST

Intrusion Detection in Depth

GCIA Practical Assignment

Version 3.3

James D. Rauser

SANS San Diego, CA

March 7- March 12, 2003

Table of Contents

Table of Contents	2
Assignment 1 Describe the State of Intrusion Detection	4
An Analysis of ARP Spoofing Detection and its Countermeasures	4
Abstract	4
Introduction	4
Functions/Capabilities	5
Detection	5
Attack Scenario 1	7
Attack Scenario 2	9
Countermeasures	14
Host Based	14
Network Based	23
References	27
Assignment 2 Network Detects	28
1 Network Detect webdav search access	28
1.1 Source of Trace:	28
1.2 Detect Generated by:	29
1.3 Probability the Source Address was Spoofed:	31
1.4 Description of Attack:	31
1.5 Attack Mechanism:	32
1.6 Correlations:	33
1.7 Evidence of Active Targeting:	36
1.8 Severity:	36
1.9 Defensive Recommendation:	36
1.10 Multiple Choice Question	37
2 Network Detect BAD TRAFFIC loopback traffic	37
2.1 Source of Trace:	37
2.2 Detect Generated by:	38
2.3 Probability the Source Address was Spoofed:	40
2.4 Description of Attack:	40
2.5 Attack Mechanism:	41
2.6 Correlations:	43
2.7 Evidence of Active Targeting:	45
2.8 Severity:	45
2.9 Defensive Recommendation:	45
2.10 Multiple Choice Question	45
3 Network Detect (spp_portscan2) Portscan detected	46
3.1 Source of Trace:	46
3.2 Detect Generated by:	47
3.3 Probability the Source Address was Spoofed:	50

3.4	Description of Attack:	50
3.5	Attack Mechanism:	51
3.6	Correlations:	51
3.7	Evidence of Active Targeting:	53
3.8	Severity:	53
3.9	Defensive Recommendation:	53
3.10	Multiple Choice Question:	54
	Three Questions from Post	54
	References	56
	Assignment 3 Analyze This	57
	Executive Summary	57
	List of Files Analyzed	58
	Analysis Process	58
	List of Detects - Description, Analysis, Correlations, Defensive Recommendations	60
	Top Talkers	103
	External Source Addresses and Registration Info	105
	Link Graph	Error! Bookmark not defined.
	References	113
	Appendix	117
	logsnorter-0.2	129
	process_scanalert.pl	159

Assignment 1 Describe the State of Intrusion Detection

An Analysis of ARP Spoofing Detection and its Countermeasures

Abstract

Given the wide availability of tools for performing ARP spoofing and since ARP spoofing is the underpinning upon which many other attacks are built upon, this paper will revisit the technique of using ARP spoofing to redirect traffic on LAN's. An assortment of countermeasures will be discussed with the primary emphasis on detection and prevention. Dug Song's Arpspoof will be demonstrated and its functions, capabilities, and limitations will be described. There are many tools available for ARP spoofing. Some others are arpoison, THC-parasite, taranis, and arptool. Ettercap, hunt and arpmitm are session hijacking tools that also use this technique.

Arpspoof <http://naughty.monkey.org/~dugsong/dsniff/>
Arpwatch <http://www.securityfocus.com/tools/142>
Snort <http://www.snort.org>

Introduction

Arpspoof is part of suite of tools associated with Dsniff¹. It is the foundation, upon which many of the other tools in the suite are built on. It allows manipulation by an attacker of the Address Resolution Protocol (ARP). ARP is used to map layer 3 addresses (32-bit IP address) to layer 2 addresses (48-bit MAC addresses). It is these mapping's, that allow traffic to be delivered to their destination. When an Ethernet frame is sent from one host on a LAN to another, it is the 48-bit MAC address that determines which interface the frame is sent to. These mappings happen without human intervention, and are usually not a concern for administrators. By changing these mappings, it is possible for an attacker to redirect traffic for interception in a switched environment².

These mappings are typically stored on routers, switches, and hosts in caches. These caches maintain the recent mappings, from ip address to hardware address³. Most systems have a timeout and the entry is removed. Ethernet ARP has four types of messages, of which we will only be concerned with two.

ARP request – a request for the destination hardware address
ARP reply – tells the requesting host the hardware address of the destination host

The weakness in the ARP protocol, on many hosts, is that it will accept unsolicited ARP replies from any host. Arpspoof can be used to redirect packets, from a target host and/or router on a LAN intended for another host, by forging ARP replies. IP Forwarding must be turned on, at the attacking host, or the attacker becomes a black hole for the victim's packets

Functions/Capabilities

The attacking host first sends out forged ARP reply packets to the target system. These forged packets, tell the victim, that the default gateway has been changed. The attacker will determine the default gateway, and replace it with his/her IP address. This assumes that the attacker, and the victim are on the same subnet with a common gateway.

In other words: the attacker continuously sends the victim computer ARP replies, containing the IP address of the gateway, and the attacker's hardware address. After some time, the victim computer will usually create an incorrect entry in his ARP cache. The next time the victim wants to send an IP packet, to the gateway, he/she sends the ethernet frame to the attacker's hardware address, so actually he/she receives the IP packet.

Now packets destined for other subnets will be sent to the attacker. To get the return packets, the attacker will send a forged packet to the gateway, claiming to be the victim. If IP forwarding has been enabled on the attacking host, this host has now in essence, become the victims default gateway, most likely without their knowledge.

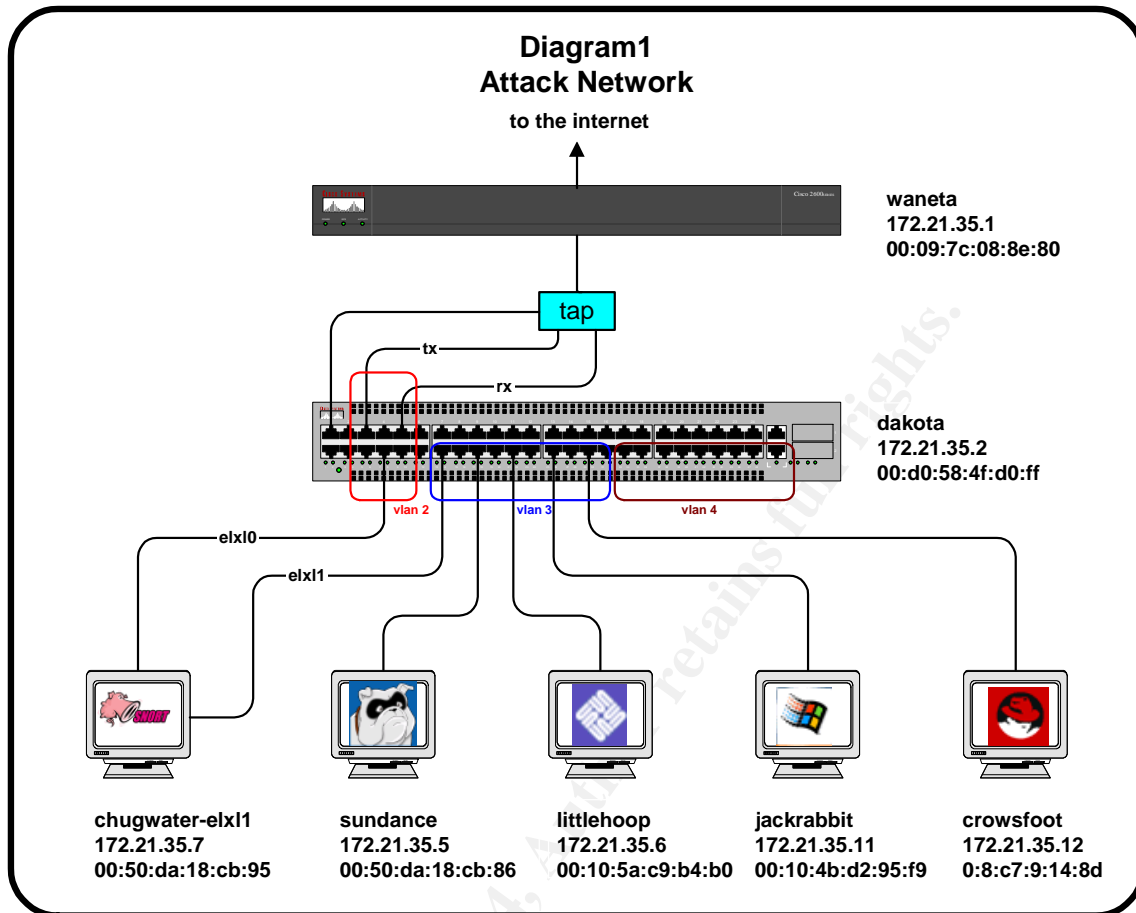
Detection

For the detection part two tools were used. Arpwatch and Snort with the arpspoof preprocessor enabled. In order for Snort to detect ARP spoofing, it must be started with the `-a` option. This will display ARP packets. The arpspoof preprocessor needs to be enabled, in the snort.conf configuration file with the following command.

preprocessor arpspoof

The detection sensor chugwater, started Snort and Arpwatch with the following commands:

```
chugwater:[/]#snort -c /usr/local/snort-1.9.1/etc/snort.conf -i eth0 -Dedba -l /var/log/elx0
chugwater:[/]#arpwatch -Dn
```



The following paragraphs describe the snort arpspoof preprocessor. They are excerpted from a reply by Jeff Nathan⁴ to a post to the snort-users mailing list.

“spp_arpspoof has 4 detection mechanisms.

(The following two methods address the use of 'preprocessor arpspoof' in snort.conf)

First: If an ARP request is observed, the source hardware address in the ethernet frame is compared to the sender Ethernet address in the ARP packet. If there is a mismatch an alert is generated.

Second: If an ARP reply is observed, the source hardware address in the Ethernet frame is compared to the sender Ethernet address in the ARP packet. Also, the destination hardware address in the Ethernet frame, is compared to the target Ethernet address within the ARP packet. If there is a mismatch in either of the two pairs of fields compared, an alert is generated.

(The following method addresses the use of 'preprocessor arpspoof: -unicast' in snort.conf)

Third: If an ARP request is observed where the destination Ethernet address in the Ethernet header is not the broadcast address (FF:FF:FF:FF:FF:FF), an alert is generated.

(The following method addresses the use of 'arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00' in snort.conf)

Fourth: A list of IP address/MAC address pairs is created in memory.

If the sender's IP address within the ARP frame matches an entry in snort's list, the MAC address in snort's list is compared to fields within the Ethernet header and ARP request packet. If either the source Ethernet address within the Ethernet header or the sender Ethernet address within the ARP packet does not match the entry in snort's list, an alert is generated. This test is performed on both ARP requests and replies."

Attack Scenario 1

The attacking host is sundance, the attacker will attempt to redirect all outbound traffic from the victim subnet (default gateway waneta) to sundance, with the following command:

```
sundance:[/]}#arpspoof 172.21.35.1
```

After the command is issued, the host sends a continuous stream of ARP replies, to the broadcast address.

```
sundance:[/]}#arpspoof 172.21.35.1
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
...
```

This is the snoop output from sundance, showing the spoofed packet. The Sender's hardware address is spoofed using sundance's hardware address of 0:50:da:18:cb:86. Also the Sender's protocol address is spoofed, since the packet comes from sundance.

ETHER: ----- Ether Header -----
ETHER:

ETHER: Packet 1 arrived at 11:08:55.63
ETHER: Packet size = 60 bytes
ETHER: Destination = ff:ff:ff:ff:ff:ff, (broadcast)
ETHER: **Source = 0:50:da:18:cb:86**,
ETHER: Ethertype = 0806 (ARP)
ETHER:
ARP: ----- ARP/RARP Frame -----
ARP:
ARP: Hardware type = 1
ARP: Protocol type = 0800 (IP)
ARP: Length of hardware address = 6 bytes
ARP: Length of protocol address = 4 bytes
ARP: Opcode 2 (ARP Reply)
ARP: **Sender's hardware address = 0:50:da:18:cb:86**
ARP: **Sender's protocol address = 172.21.35.1, waneta.bepc.net**
ARP: Target hardware address = ff:ff:ff:ff:ff:ff
ARP: Target protocol address = 0.0.0.0, OLD-BROADCAST
ARP:

A quick check on a victim machine's, ARP cache's shows a successful attack. All the hosts littlehoop (Solaris 8 x86), crowsfoot (Linux 7.2), and jackrabbit (Windows 2000) caches were updated to the forged hardware address. After that, all of littlehoop's, crowsfoot's and jackrabbit's outbound traffic, was sent to Sundance, before forwarding to waneta (the true default gateway).

The following excerpts show the forged entries in littlehoop, crowsfoot and jackrabbit's ARP cache.

```
littlehoop:[/]#arp -a
Net to Media Table: IPv4
Device  IP Address          Mask    Flags  Phys Addr
-----
elxl0  waneta.bepc.net      255.255.255.255    00:50:DA:18:CB:86

[root@crowsfoot root]# arp -a
waneta.bepc.net (172.21.35.1) at 00:50:DA:18:CB:86 [ether] on eth0

jackrabbit's prompt
C:\>arp -a
Interface: 172.21.35.11 on Interface 2
Internet Address    Physical Address    Type
172.21.35.1        00-50-DA-18-CB-86  dynamic
```

This is detected by Snort on chugwater, with the following alert.

[**] [112:3:1] (spp_arpspoof) Ethernet/ARP Mismatch request for Destination [**]
04/03-11:08:57.632164

Arpwatch reports it as:

From: arpwatch (Arpwatch)
To: root
Subject: flip flop

hostname: waneta.bepc.net
ip address: 172.21.35.1
ethernet address: 0:50:da:18:cb:86
ethernet vendor: 3COM CORPORATION
old ethernet address: 0:9:7c:8:8e:80
old ethernet vendor: <unknown>
timestamp: Thursday, April 3, 2003 11:09:04 -0600
previous timestamp: Thursday, April 3, 2003 11:09:02 -0600
delta: 2 seconds

Attack Scenario 2

In this attack scenario, the attacker (sundance) will go after one host (crowsfoot) and attempt to capture traffic in both directions as follows. The following line was added, to the snort configuration file, snort.conf: This is the gateways ip and mac address's.

```
preprocessor arpspoof_detect_host: 172.21.35.1 00:09:7c:08:8e:80
```

Snort was killed and restarted. Arpwatch was also.

Tell the victim host that we are the gateway
arpspoof -t victim gateway

Tell the gateway that we are the victim.
arpspoof -t gateway victim

The actual commands are as follows, which sends a continuous stream of ARP replies.

```
sundance:[/#]#arpspoof -t 172.21.35.1 172.21.35.12  
0:50:da:18:cb:86 0:9:7c:8:8e:80 0806 42: arp reply 172.21.35.12 is-at  
0:50:da:18:cb:86  
0:50:da:18:cb:86 0:9:7c:8:8e:80 0806 42: arp reply 172.21.35.12 is-at  
0:50:da:18:cb:86  
...
```

from another shell

```
sundance:[/#]#arpspoof -t 172.21.35.12 172.21.35.1
```

```
0:50:da:18:cb:86 0:8:c7:9:14:8d 0806 42: arp reply 172.21.35.1 is-at  
0:50:da:18:cb:86  
0:50:da:18:cb:86 0:8:c7:9:14:8d 0806 42: arp reply 172.21.35.1 is-at  
0:50:da:18:cb:86  
...
```

This is the snoop output from sundance, showing a sample spoofed packet. The Sender's hardware address is spoofed using sundance's hardware address of 0:50:da:18:cb:8. Also the Sender's protocol address is spoofed, since the packet comes from sundance.

```
ETHER: ----- Ether Header -----  
ETHER:  
ETHER: Packet 34 arrived at 18:07:55.39  
ETHER: Packet size = 60 bytes  
ETHER: Destination = 0:9:7c:8:8e:80,  
ETHER: Source = 0:50:da:18:cb:86,  
ETHER: Ethertype = 0806 (ARP)  
ETHER:  
ARP: ----- ARP/RARP Frame -----  
ARP:  
ARP: Hardware type = 1  
ARP: Protocol type = 0800 (IP)  
ARP: Length of hardware address = 6 bytes  
ARP: Length of protocol address = 4 bytes  
ARP: Opcode 2 (ARP Reply)  
ARP: Sender's hardware address = 0:50:da:18:cb:86  
ARP: Sender's protocol address = 172.21.35.12, crowfoot.bepc.net  
ARP: Target hardware address = 0:9:7c:8:8e:80  
ARP: Target protocol address = 172.21.35.1, waneta.bepc.net  
ARP:
```

```
ETHER: ----- Ether Header -----  
ETHER:  
ETHER: Packet 2 arrived at 18:10:21.14  
ETHER: Packet size = 60 bytes  
ETHER: Destination = 0:8:c7:9:14:8d,  
ETHER: Source = 0:50:da:18:cb:86,  
ETHER: Ethertype = 0806 (ARP)  
ETHER:  
ARP: ----- ARP/RARP Frame -----  
ARP:  
ARP: Hardware type = 1  
ARP: Protocol type = 0800 (IP)  
ARP: Length of hardware address = 6 bytes  
ARP: Length of protocol address = 4 bytes
```

ARP: Opcode 2 (ARP Reply)
ARP: **Sender's hardware address = 0:50:da:18:cb:86**
ARP: **Sender's protocol address = 172.21.35.1, waneta.bepc.net**
ARP: Target hardware address = 0:8:c7:9:14:8d
ARP: Target protocol address = 172.21.35.12, crowsfoot.bepc.net
ARP:

A quick check on a victim machine's ARP cache, shows the forged entry was successful.

```
[root@crowsfoot root]# arp -a  
waneta.bepc.net (172.21.35.1) at 00:50:DA:18:CB:86 [ether] on eth0
```

The router (waneta) shows a successful attack (crowsfoot's ip address with sundance's mac address).

```
waneta#sh arp  
Protocol Address      Age (min) Hardware Addr  Type   Interface  
Internet 172.21.35.12    0    0050.da18.cb86  ARPA   FastEthernet0/13  
Internet 172.21.35.2    15    00d0.584f.d0ff  ARPA   FastEthernet0/13  
Internet 172.21.35.1      -    0009.7c08.8e80  ARPA   FastEthernet0/13  
Internet 172.21.35.6     10    0010.5ac9.b4b0  ARPA   FastEthernet0/13  
Internet 172.21.35.7      2    0050.da18.cb97  ARPA   FastEthernet0/13  
Internet 172.21.35.5      8    0050.da18.cb86  ARPA   FastEthernet0/13
```

This is detected by Snort on chugwater, with the following alert.

```
[**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [**]  
04/05-18:08:11.648786
```

Arpwatch reports it as:

From: arpwatch (Arpwatch)
To: root
Subject: flip flop (crowsfoot.bepc.net)

hostname: crowsfoot.bepc.net
ip address: 172.21.35.12
ethernet address: 0:50:da:18:cb:86
ethernet vendor: 3COM CORPORATION
old ethernet address: 0:8:c7:9:14:8d
old ethernet vendor: COMPAQ COMPUTER CORPORATION
timestamp: Saturday, April 5, 2003 18:09:05 -0600
previous timestamp: Saturday, April 5, 2003 14:16:25 -0600
delta: 3 hours

From: arpwatch (Arpwatch)
To: root
Subject: flip flop (waneta.bepc.net)

hostname: waneta.bepc.net
ip address: 172.21.35.1
ethernet address: 0:50:da:18:cb:86
ethernet vendor: 3COM CORPORATION
old ethernet address: 0:9:7c:8:8e:80
old ethernet vendor: <unknown>
timestamp: Saturday, April 5, 2003 18:20:25 -0600
previous timestamp: Saturday, April 5, 2003 18:19:13 -0600
delta: 1 minute

From: arpwatch (Arpwatch)
To: root
Subject: flip flop (waneta.bepc.net)

hostname: waneta.bepc.net
ip address: 172.21.35.1
ethernet address: 0:9:7c:8:8e:80
ethernet vendor: <unknown>
old ethernet address: 0:50:da:18:cb:86
old ethernet vendor: 3COM CORPORATION
timestamp: Saturday, April 5, 2003 18:21:17 -0600
previous timestamp: Saturday, April 5, 2003 18:21:17 -0600
delta: 0 seconds

From: arpwatch (Arpwatch)
To: root
Subject: flip flop (waneta.bepc.net)

hostname: waneta.bepc.net
ip address: 172.21.35.1
ethernet address: 0:50:da:18:cb:86
ethernet vendor: 3COM CORPORATION
old ethernet address: 0:9:7c:8:8e:80
old ethernet vendor: <unknown>
timestamp: Saturday, April 5, 2003 18:21:19 -0600
previous timestamp: Saturday, April 5, 2003 18:21:17 -0600
delta: 2 seconds

From: arpwatch (Arpwatch)
To: root
Subject: flip flop (waneta.bepc.net)

hostname: waneta.bepc.net
ip address: 172.21.35.1
ethernet address: 0:9:7c:8:8e:80
ethernet vendor: <unknown>
old ethernet address: 0:50:da:18:cb:86
old ethernet vendor: 3COM CORPORATION
timestamp: Saturday, April 5, 2003 18:23:29 -0600
previous timestamp: Saturday, April 5, 2003 18:23:29 -0600
delta: 0 seconds

these continue flopping back and forth with delta's of 0 to 2 seconds until it finally settles down.

Interestingly enough when a ping is done from crowsfoot , pinging deadhorse (a host on another subnet) we get redirects from sundance (the attacker). Obviously the attacker is not that skilled at setting up a stealthy attack.

```
[root@crowsfoot root]# ping deadhorse
PING deadhorse.bepc.net (172.21.51.174) from 172.21.35.12 : 56(84) bytes of
data.
From sundance.bepc.net (172.21.35.5): Redirect Host(New nexthop:
waneta.bepc.net (172.21.35.1))
64 bytes from deadhorse.bepc.net (172.21.51.174): icmp_seq=0 ttl=252
time=3.395 msec
....
```

Snort also alerts on this as follows, this could also be another indicator of an attempted ARP spoofing attack on your LAN. Although, probably by an unsophisticated attacker. The attacker is on a Solaris host and the following command would stop the redirects: `ndd -set /dev/ip ip_send_redirects 0`. The snort alert is shown below.

```
[**] [1:472:1] ICMP redirect host [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
04/05-18:29:13.756085 0:50:DA:18:CB:86 -> 0:8:C7:9:14:8D type:0x800
len:0x5A
172.21.35.5 -> 172.21.35.12 ICMP TTL:255 TOS:0x0 ID:9959 IpLen:20
DgmLen:76 DF
Type:5 Code:1 REDIRECT HOST NEW GW: 172.21.35.1
** ORIGINAL DATAGRAM DUMP:
172.21.35.12:22 -> 172.21.27.6:1321 TCP TTL:64 TOS:0x0 ID:0 IpLen:20
DgmLen:48 DF
***A**S* Seq: 0x40C794E Ack: 0x1E9B58C5 Win: 0x16D0 TcpLen: 28
** END OF DUMP
[Xref => cve CVE-1999-0265][Xref => arachnids 135]
Original packet from snort.
```

```
04/05-18:29:13.756085 172.21.35.5 -> 172.21.35.12
ICMP TTL:255 TOS:0x0 ID:9959 IpLen:20 DgmLen:76 DF
Type:5 Code:1 REDIRECT HOST NEW GW: 172.21.35.1
** ORIGINAL DATAGRAM DUMP:
172.21.35.12:22 -> 172.21.27.6:1321
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
***A**S* Seq: 0x40C794E Ack: 0x1E9B58C5 Win: 0x16D0 TcpLen: 28
** END OF DUMP
AC 15 23 01 45 00 00 30 00 00 40 00 40 06 A4 8B ..#.E..0..@.@...
AC 15 23 0C AC 15 1B 06 00 16 05 29 04 0C 79 4E ..#.....).yN
1E 9B 58 C5 70 12 16 D0 DC 08 00 00 02 04 05 B4 ..X.p.....
01 01 04 02
```

Countermeasures

Host Based

It is very difficult to defend against ARP attacks due to the underlying weaknesses in the protocol. Controlling the behavior of the ARP cache may give some protection. Although, it may be difficult to implement in large environments, and may cause unintended side effects, one supposed defense against ARP attacks is to reduce the lifetime of cache entries. However an ARP time-out that is too short may produce troubles, particularly on large networks. Hosts that are constantly dumping their ARP caches due to short time-out values will cause more broadcasting. This will have a direct, negative impact on performance, since the IP software will not be able to send any data until an ARP broadcast has been sent and responded to. On the other hand, ARP cache timeouts that are too high might cause problems. For example, whenever a host is assigned a different IP address, the other hosts who have an older entry in their caches will still try to send data to the old (and invalid) hardware address.

Another option could be to create static ARP addresses for some systems. Static ARP cache entries are permanent and therefore do not expire. These entries can be deleted using the command `arp -d`. A third option would be to disable ARP processing on the system interface(s) altogether and add static ARP entries.

Windows 2000 Professional

The following commands displays and modifies the IP-to-physical address translation tables used by the Address Resolution Protocol (ARP).

```
arp -a [inet_addr] [-N [if_addr]]
```

arp -d *inet_addr* [*if_addr*]

arp -s *inet_addr ether_addr* [*if_addr*]

The **-s** option adds an entry in the ARP cache to associate the IP address *inet_addr* with the physical address *ether_addr*. The physical address is given as 6 hexadecimal bytes separated by hyphens. The IP address is specified using dotted decimal notation. The entry is permanent, that is, it is not automatically removed from the cache after the time-out expires. The **-a** option displays the current entries in the cache and the **-d** option deletes entries from the cache.

To minimize ARP broadcast traffic on your network, Windows 2000 maintains a cache of hardware-to-software address mappings for future use. This cache contains the following two types of entries, dynamic and static.

Dynamic ARP cache entries are added and deleted automatically during the normal use of TCP/IP sessions with remote computers. Dynamic entries age and expire from the cache if not reused within 2 minutes. Windows 2000 adjusts the size of the ARP cache automatically to meet the needs of the system. If an entry is not used by any outgoing datagram for two minutes, the entry is removed from the ARP cache. Entries that are being referenced are given additional time, in two minute increments, up to a maximum lifetime of 10 minutes. After 10 minutes, the ARP cache entry is removed and must be rediscovered using an ARP Request frame. To adjust the time an unreferenced entry can remain in the ARP cache, change the value of the `ArpCacheLife` and/ or the `ArpCacheMinReferencedLife` registry entries.

Static ARP cache entries are added manually by using the ARP command with the **-s** option. Static entries remain in the ARP cache until the computer is restarted⁵.

The ARP cache is erased upon initialization of the TCP/IP protocol. To make static ARP cache entries persistent, each time the computer is started, create a command file with the ARP commands and place a shortcut to the command file in the Startup folder.

In addition to creating an ARP cache entry through the receipt of an ARP Reply, ARP cache entries are updated if the mapping is received through an ARP Request. In other words, if the IP address of the sender of an ARP Request is in the cache, update the entry with the sender's MAC address. This way, nodes that have static or dynamic ARP cache entries for the sender are updated with the ARP Request sender's current MAC address. For a node whose interface and MAC address changes, it updates the ARP cache containing an entry for the node the next time the node sends an ARP Request.

So lets attempt to overwrite the ARP cache as shown below. First, checking the current contents.

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address      Physical Address      Type
  172.21.35.1          00-09-7c-08-8e-80    dynamic
C:\>
```

Then start the attack from sundance and attempt to overwrite the entry for the default gateway on jackrabbit.

```
sundance:[/]/#arp spoof -t 172.21.35.1 172.21.35.1
0:50:da:18:cb:86 0:10:4b:d2:95:f9 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 0:10:4b:d2:95:f9 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 0:10:4b:d2:95:f9 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
...
```

```
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address      Physical Address      Type
  172.21.35.1          00-50-da-18-cb-86    dynamic
  172.21.35.5          00-50-da-18-cb-86    dynamic
```

it is immediately overwritten and when the attack stops the correct entry is once again found in the cache

```
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address      Physical Address      Type
  172.21.35.1          00-09-7c-08-8e-80    dynamic
  172.21.35.5          00-50-da-18-cb-86    dynamic
```

Now lets attempt to stop the attack by adding a static entry for the gateway to the cache. To change TCP/IP settings, you must be logged on as a member of the Administrator group. This is by design or we get a message like the following.

```
C:\>arp -s 172.21.35.1 00-09-7c-08-8e-80
```

The ARP entry addition failed: 5

So after switching to an administrator account we attempt this again.

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address   Physical Address   Type
  172.21.35.1       00-09-7c-08-8e-80 dynamic
C:\>arp -s 172.21.35.1 00-09-7c-08-8e-80
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address   Physical Address   Type
  172.21.35.1       00-09-7c-08-8e-80 static
```

Starting the attack again, and inspecting the ARP cache shows that it is overwritten once again. So this is not a successful counter measure for this platform.

```
C:\>arp -a
Interface: 172.21.35.11 on Interface 0x10000003
  Internet Address   Physical Address   Type
  172.21.35.1       00-50-da-18-cb-86 static
  172.21.35.5       00-50-da-18-cb-86 dynamic
```

Now lets turn our attention to the ARP cache timeout parameters. Windows NT and Windows 2000 adjust the size of the ARP cache automatically to meet the needs of the system. If an entry is not used by any outgoing datagram for two minutes, the entry is removed from the ARP cache. Entries that are being referenced are removed from the ARP cache after ten minutes. Entries added manually are not removed from the cache automatically. A new registry parameter, ArpCacheLife, was added in Windows NT 3.51 Service Pack 4 to allow more administrative control over aging.

All of the TCP/IP parameters are registry values located under the registry following Registry path:

```
HKEY_LOCAL_MACHINE
  \SYSTEM
    \CurrentControlSet
      \Services:
        \Tcpip
          \Parameters
```

ArpCacheLife REG_DWORD Number of Seconds

Default: 600 (10 minutes) In the absence of an ArpCacheLife parameter, the defaults for ARP cache time-outs are a two-minute time-out on unused entries and a ten-minute time-out on used entries.

Description: Determines the default lifetime for entries in the ARP cache table. Once an entry is placed in the ARP cache, it is allowed to remain there until its lifetime expires or until its table entry is reused because it is the oldest entry. If ArpCacheLife is greater than or equal to ArpCacheMinReferencedLife, referenced and un-referenced ARP cache entries expire in ArpCacheLife seconds.

If ArpCacheLife is less than ArpCacheMinReferencedLife, un-referenced entries expire in ArpCacheLife seconds, and referenced entries expire in ArpCacheMinReferencedLife seconds.

ArpCacheSize REG_DWORD Number

Default: 62

Determines the maximum number of entries that the ARP cache table can hold. The ARP cache is allowed to grow dynamically until this size is reached. After the table reaches this size, new entries can only be added by replacing the oldest entries that exist.

ArpCacheMinReferencedLife REG_DWORD Number of Seconds

Default: 600 seconds (10 minutes)

Description: ArpCacheMinReferencedLife controls the minimum time until a referenced ARP cache entry expires. This parameter can be used in combination with the ArpCacheLife parameter, as follows:

Entries in the ARP cache are referenced each time that an outbound packet is sent to the IP address in the entry⁶.

Adapter-specific values are listed under subkeys for each adapter. Depending on whether the system or adapter is DHCP-configured or static override values are specified, parameters may have both DHCP and statically configured values. If any of these parameters are changed using the registry editor, a reboot of the system is generally required for the change to take effect. A reboot is usually not required if values are changed using the network connections interface.

If we now experiment with the ARP cache aging parameters by setting the

arp cache life entry to 10 seconds and rebooting. Once again start an attack and the ARP cache on the victim is overwritten. So let's set the timer to 0 and reboot. Now examining the cache we see any entries are immediately timed out and flushed from the cache.

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\>arp -a
No ARP Entries Found
C:\>arp -a
No ARP Entries Found
C:\>arp -a
No ARP Entries Found
C:\>arp -a
No ARP Entries Found
```

Now the host has to send an ARP request for every packet as demonstrated by pinging a host. This should slow down an attacker. So let's start an attack from sundance telling jackrabbit we are the default gateway. Then on the victim (jackrabbit) generate some traffic by pinging a host on another subnet. We will capture the ARP traffic on chugwater.

```
C:\>ping deadhorse -t
Pinging deadhorse.bepc.net [172.21.51.174] with 32 bytes of data:
Reply from 172.21.51.174: bytes=32 time<10ms TTL=253
Reply from 172.21.51.174: bytes=32 time<10ms TTL=253
...
```

which generates constant ARP requests from jackrabbit (for the default gateway). This is shown on a snoop from chugwater.

```
jackrabbit.bepc.net -> (broadcast) ARP C Who is 172.21.35.1, waneta.bepc.net
?
waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is
0:9:7c:8:8e:80
waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is
0:50:da:18:cb:86
jackrabbit.bepc.net -> (broadcast) ARP C Who is 172.21.35.1, waneta.bepc.net
?
waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is
0:9:7c:8:8e:80
waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is
0:50:da:18:cb:86
jackrabbit.bepc.net -> (broadcast) ARP C Who is 172.21.35.1, waneta.bepc.net
?
```

waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is 0:9:7c:8:8e:80

waneta.bepc.net -> jackrabbit.bepc.net ARP R 172.21.35.1, waneta.bepc.net is 0:50:da:18:cb:86

The cache is initially empty, until the attack starts then it is overwritten. Occasionally the cache empties out, but is immediately overwritten. Sometimes waneta's true value is in there but not very often. The cache, is overwritten by the attacker, most of the time, so most of the packets go to the attacker.

I started a snoop on sundance looking for icmp then started a ping with a count option on jackrabbit. Almost invariably the attacker (sundance) received the packet. Only occasionally, it did not. Given that, it is constantly sending ARP replies, the odds are, the ARP reply from sundance is being received before waneta's, so sundance updates the cache most of the time.

Another trace upholds this idea, a constant ping was started on jackrabbit and another snoop was started on chugwater. In that snoop, sundance is seen sending approximately six replies to waneta's one. Adjusting the ARP cache parameters is not an effective countermeasure on this platform, as it also increases traffic and degrades performance.

Solaris 8 x86

The cache lifetime is determined by the kernel parameter `arp_cleanup_interval`. The IP routing table entry lifetime is controlled by the kernel parameter `ip_ire_arp_interval`. On Solaris hosts we can set the ARP timers to lower values⁷.

```
# ndd -set /dev/arp arp_cleanup_interval <time>
# ndd -set /dev/ip ip_ire_arp_interval <time>
```

where <time> is in milliseconds. Reducing the ARP cache timeout interval and the IP-routing table timeout interval will slow down an attacker but not stop them.

The `arp_cleanup_interval` option determines the period of time the Address Resolution Protocol (ARP) cache maintains entries. ARP attacks are still effective with the default interval. Shortening the timeout interval should reduce the effectiveness of such an attack. The default value is 300000 milliseconds (5 minutes).

The `ip_ire_arp_interval` option determines the period of time at which a specific route will be kept, even if currently in use. ARP attacks may be effective with the default interval. Shortening the time interval may reduce the effectiveness of attacks. The default interval is 1200000 milliseconds (20 minutes)⁸.

Get current littlehoop entries and make the following changes, as shown below.

```
littlehoop:[/] ndd -get /dev/arp arp_cleanup_interval
300000
littlehoop:[/] ndd -get /dev/ip ip_ire_arp_interval
1200000

Discard ARP entries from ARP cache after 1 minute.
littlehoop:[/] ndd -set /dev/arp arp_cleanup_interval 60000

Flush ARP entries from routing table after 1 minute
littlehoop:[/] ndd -set /dev/ip ip_ire_arp_interval 60000
```

Changing the above timers had little effect. I was able to ARP spoof and have the entry immediately added to the victims ARP cache. As shown below:

```
littlehoop:[/]#arp -a
Net to Media Table: IPv4
Device  IP Address          Mask    Flags  Phys Addr
-----
elxl0   waneta.bepc.net       255.255.255.255    00:50:da:18:cb:86
elxl0   littlehoop            255.255.255.255 SP  00:10:5a:c9:b4:b0
elxl0   base-address.mcast.net 240.0.0.0    SM  01:00:5e:00:00:00
littlehoop:[/]#
```

Next I added a static static entry for router, in littlehoop's arp cache

```
littlehoop:[/]#arp -s waneta.bepc.net 00:09:7c:08:8e:80
littlehoop:[/]#arp -a
Net to Media Table: IPv4
Device  IP Address          Mask    Flags  Phys Addr
-----
elxl0   littlehoop            255.255.255.255 SP  00:10:5a:c9:b4:b0
elxl0   waneta.bepc.net       255.255.255.255 S   00:09:7c:08:8e:80
elxl0   base-address.mcast.net 240.0.0.0    SM  01:00:5e:00:00:00
```

After running the ARP spoof attack the entry was immediately overwritten.

```
littlehoop:[/]#arp -a
Net to Media Table: IPv4
Device  IP Address          Mask    Flags  Phys Addr
-----
elxl0   waneta.bepc.net       255.255.255.255 S   00:50:da:18:cb:86
elxl0   littlehoop            255.255.255.255 SP  00:10:5a:c9:b4:b0
```

```
elxl0 base-address.mcast.net 240.0.0.0 SM 01:00:5e:00:00:00
```

I must be missing something here! The supposed defenses are having no effect, whatsoever! They are not stopping the attack or slowing it down. The cache is being overwritten. Apparently the cache is being cleaned every minute, but since is being immediately spoofed with forged entries, the supposed defense is having very little effect.

So now, lets try this, reduce the cache interval some more, lets discard the ARP entries after ten seconds.

```
littlehoop:[/] ndd -set /dev/arp arp_cleanup_interval 1000
```

After rebooting I am unable to overwrite after 5 minutes of trying. I stopped and started the attack multiple times. I was unable to overwrite the ARP cache any of the times even though the attacks were allowed to run for minutes. Apparently, the timers have to be set to a lower value for this to be an effective defense.

Cisco 3550

The router is a Cisco 3550, after setting the router interface ARP timeout, to 10 seconds, with the following commands. I am still able to spoof the router.

```
dakota(config)#int vlan 3
dakota(config-if)#arp timeout 10
dakota#sh arp
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	172.21.35.11	0	0010.4bd2.95f9	ARPA	Vlan3
Internet	172.21.35.8	0	0050.da18.cb86	ARPA	Vlan3
Internet	172.21.35.12	0	0008.c709.148d	ARPA	Vlan3
Internet	172.21.35.2	0	00d0.584f.d0ff	ARPA	Vlan3
Internet	172.21.35.1	-	0009.7c08.8e80	ARPA	Vlan3
Internet	172.21.35.7	0	0050.da18.cb95	ARPA	Vlan3
Internet	172.21.35.5	0	0050.da18.cb86	ARPA	Vlan3

So then I added added a static entry for littlehoop (the victim).

```
dakota(config)#arp 172.21.35.8 0010.5ac9.b4b0 arpa
dakota#sh arp
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	172.21.35.8	-	0010.5ac9.b4b0	ARPA	
Internet	172.21.35.12	0	0008.c709.148d	ARPA	Vlan3
Internet	172.21.35.2	0	00d0.584f.d0ff	ARPA	Vlan3
Internet	172.21.35.1	-	0009.7c08.8e80	ARPA	Vlan3

Internet	172.21.35.7	0	0050.da18.cb95	ARPA	Vlan3
Internet	172.21.35.5	0	0050.da18.cb86	ARPA	Vlan3

This put a stop to spoofing the routers ARP cache. This is an effective countermeasure for small LANs.

Redhat 7.2

After adding the static entry for the gateway, when I try to ARP spoof crowsfoot from sundance I cannot overwrite the ARP cache. Without the static entry I can overwrite it immediately. This countermeasure works.

```
[root@crowsfoot root]# arp -s waneta.bepc.net 00:09:7c:08:8e:80
[root@crowsfoot root]# arp -a
waneta.bepc.net (172.21.35.1) at 00:09:7C:08:8E:80 [ether] PERM on eth0
[root@crowsfoot root]#
```

Network Based

VLANS

Virtual LANs offer some level of protection, for hosts that are not on the same VLAN as an attacker. If we attempt to ARP spoof, crowsfoot on VLAN 4, from sundance on VLAN 3 using the following commands. Arpspoof needs crowsfoot's mac address so it sends an ARP request, and can't get it across the vlan. Similarly if we try to tell crowsfoot we are the default gateway it sends an ARP request for crowsfoot, which it can't find.

```
sundance:[/]# arpspoof -t 172.21.35.33 172.21.35.40
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.33 tell 172.21.35.5
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.33 tell 172.21.35.5
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.33 tell 172.21.35.5
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.33 tell 172.21.35.5
couldn't arp for host 172.21.35.33
sundance:[/]#
```

or the other way

```
sundance:[/]# arpspoof -t 172.21.35.40 172.21.35.33
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.40 tell 172.21.35.5
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.40 tell 172.21.35.5
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.40 tell 172.21.35.5
```



```
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp who-has 172.21.35.40 tell 172.21.35.5
couldn't arp for host 172.21.35.40
sundance:[/]#
```

If we try to redirect all traffic on the subnet, we are able to send the forged ARP replies, this is because all VLANs use the same mac address as shown below.

```
sundance:[/]#arp spoof 172.21.35.33
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.33 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.33 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 ff:ff:ff:ff:ff:ff 0806 42: arp reply 172.21.35.33 is-at
0:50:da:18:cb:86
0:50:da:18
```

However no entry is made in the routers ARP cache, so VLANs do protect against ARP spoofing.

```
waneta#sh arp
Protocol Address      Age (min) Hardware Addr  Type   Interface
Internet 172.21.35.249      -    0009.7c08.8e80  ARPA   Vlan2
Internet 172.21.35.1      -    0009.7c08.8e80  ARPA   Vlan3
Internet 172.21.35.7      0    0050.da18.cb95  ARPA   Vlan3
Internet 172.21.35.5      2    0050.da18.cb86  ARPA   Vlan3
Internet 172.21.35.43      7    0010.5ac9.b4b0  ARPA   Vlan4
Internet 172.21.35.40      9    0008.c709.148d  ARPA   Vlan4
Internet 172.21.35.41     110   0010.4bd2.95f9  ARPA   Vlan4
Internet 172.21.35.33      -    0009.7c08.8e80  ARPA   Vlan4
```

Port Based Security

Richard Duffy stated in his paper⁹ that he was unable to keep arpspoof from spoofing the router. With the newer security features in switches this is now possible. Assuming a switched environment, switches may be locked down by MAC address.

Securing ports can be tedious work. Unused ports should be disabled. Other methods involve MAC address hardcoding. Key devices such as routers and firewalls can be secured by hardcoding their MAC address into the switch configuration. For the Cisco CatOS the following command on a Catalyst 4000 series

```
dakota> (enable) set port security 2/32 disable age 0 maximum 1 shutdown 10
```

violation shutdown

has the following effect. The age time specifies how long the address will be secure. Setting it to 0 disables aging. The maximum indicates the number of MAC addresses allowed. The security violation action can be specified as either shutdown or restrict. Shutdown shuts down the port permanently for a specified time in minutes. The valid range is 10 to 1440 minutes. If it is set to zero the shutdown is disabled for that port. When the shutdown timeout expires, the port is re-enabled and all port security configurations are maintained. Restrict drops all packets from insecure hosts but remains enabled.

For the example, on our switch dakota, if we attempt to set port based security of the router port with the following command.

```
dakota> (enable) set port security 2/3 enable violation restrict
Feature not allowed on trunking port.
dakota> (enable)
```

We can't use this command on a trunk port. So if we set the host port as follows, setting the port to 1 MAC address with a violation action to shutdown for 10 minutes.

```
dakota> (enable) set port security 2/14 enable violation shutdown maximum 1
age 10
Port 2/14 security enabled, maximum address 1, age time 10, violation mode
shutdown.
Trunking disabled for Port 2/14 due to Security Mode.
dakota> (enable)
```

If we now start an ARP spoofing attack telling littlehoop we are the default gateway.

```
sundance:[/]/#arp spoof -t 172.21.35.8 172.21.35.1
0:50:da:18:cb:86 0:10:5a:c9:b4:b0 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 0:10:5a:c9:b4:b0 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
```

The switch sees the spoofed packets and generates a warning.

```
dakota> (enable) 2003 Apr 13 21:12:44 cst -05:00 %SYS-4-P2_WARN: 1/Traffic
from permanent host 00:10:5a:c9:b4:b0 but seen on incorrect port 2/14
```

When the arpspoof command is stopped, the switch issues the following warning. Since it is seeing ARP replies for the same host from different sources.

```
2003 Apr 13 21:14:26 cst -05:00 %SYS-4-P2_WARN: 1/Host 00:09:7c:08:8e:80
is flapping between port 2/14 and port 2/3
```

Start it again and let it run this time.

```
sundance:[/ ]#arp spoof -t 172.21.35.8 172.21.35.1
0:50:da:18:cb:86 0:10:5a:c9:b4:b0 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 0:10:5a:c9:b4:b0 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
0:50:da:18:cb:86 0:10:5a:c9:b4:b0 0806 42: arp reply 172.21.35.1 is-at
0:50:da:18:cb:86
```

It takes a couple of minutes for the switch to respond with the following.

```
dakota> (enable) 2003 Apr 13 21:29:18 cst -05:00 %SECURITY-1-
PORTSHUTDOWN:Port 2/14 shutdown due to security violation
```

Looking at the port status we see the last source address to be that of the router.

```
dakota> (enable) sh port 2/14
```

Port	Name	Status	Vlan	Level	Duplex	Speed	Type
2/14		shutdown	3	normal	full	100	10/100BaseTX

Port	Security Violation	Shutdown-Time	Age-Time	Max-Addr	Trap	IfIndex
2/14	enabled shutdown	0	10	1	disabled	70

Port	Num-Addr	Secure-Src-Addr	Age-Left	Last-Src-Addr	Shutdown/Time-Left
2/14	0	-	-	00-09-7c-08-8e-80	yes -

Other port security settings for Cisco switches include the fairly new LEAP or layer two authentication defensive mechanisms.

References

¹ Song, Dug. “dsniff.”

URL: <http://naughty.monkey.org/~dugsong/dsniff>

² Skoudis, Ed. Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses. 2002 Prentice Hall. Pages 58-60

³ Stevens, Richard W. TCP/IP Illustrated, Volume 1 The Protocols. 1994 Addison-Wesley. Pages 54-57

⁴ Nathan, Jeff. <http://www.geocrawler.com/archives/3/4890/2002/6/0/9056309/>

⁵ Microsoft Corporation. “Microsoft Windows 2000 Server Documentation.” February 28, 2000. Microsoft Corporation.
URL: http://www.microsoft.com/windows2000/en/server/help/default.asp?url=/windows2000/en/server/help/sag_TCPIP_pro_ArpCache.htm

⁶ MacDonald, Dave and Barkley, Warren. “Microsoft Windows 2000 TCP/IP Implementation Details.” January 6, 2000. Microsoft Corporation
URL: http://www.microsoft.com/windows2000/techinfo/howitworks/communication/networkbasics/tcpip_implement.asp

⁷ Dubrawsky, Ido. “Solaris Kernel Tuning for Security.” December 20, 2000
URL: <http://www.securityfocus.com/infocus/1385>

⁸ Watson, Keith. “Nddconfig script.” 1991 Sun Microsystems, Inc.
URL: <http://www.fish.com/titan/arch/sol2sun4/lib/nddconfig>

⁹ Duffy, Richard. “Finding dsniff on your Network.” November 28, 2001
URL: <http://www.sans.org/rr/penetration/dsniff.php>

Russel, Christopher R. “Penetration Testing with dsniff.” February 18, 2001
URL: <http://www.sans.org/rr/threats/dsniff.php>

Loeb, Larry. “On the lookout for dsniff” Part 1. January 2001
URL: <http://www-106.ibm.com/developerworks/library/s-sniff.html>

Wagner, Robert. “Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks”. September 27, 2001.
URL: <http://www.sans.org/rr/threats/address.php>

Assignment 2 Network Detects

1 Network Detect webdav search access

1.1 Source of Trace:

This alert was captured on a friend's network using Snort 1.9.0 with the ruleset (\$Id: sid,v 1.92.2.1 2003/02/09 03:27:29 cazz) and is displayed below with SnortSnarf¹. The Snort sensor sniffs traffic from a tap on a screened subnet, and watches traffic bound for various application web servers. The firewall has port 80 open to the target host. The firewall is an application level proxy and does not allow URLs greater than 2048. The web server is IIS 5.0 and has WebDAV enabled and is running URL scan.

	SnortSnarf signature page <u>WEB-MISC webdav search access</u> SnortSnarf v021024.1

6 alerts with this signature using input module SnortFileInput, with sources:

- /var/log/snort-elx10/05-13-03/alert
- /var/log/snort-elx10/05-13-03/portscan.log

Earliest such alert at **00:03:01.953533** on 05/13/2003

Latest such alert at **00:05:19.297191** on 05/13/2003

WEB-MISC webdav search access	1 sources	1 destinations
Priority: 2	Classification: access to a potentially vulnerable web application	
[sid:1070] [arachNIDS:474]		
Rules with message "WEB-MISC webdav search access":		
alert tcp \$EXTERNAL_NET any -> \$HTTP_SERVERS \$HTTP_PORTS (msg:"WEB-MISC webdav search access"; flow:to_server,established; content: "SEARCH "; depth: 8; nocase;reference:arachnids,474; classtype:web-application-activity; sid:1070; rev:5;) (from <i>web-misc.rules</i>)		

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
202.109.114.237	6	6	1	1

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
xxx.xxx.xxx.xxx	6	11	1	4

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Wed May 14 13:02:24 2003

1.2 Detect Generated by:

The alert was generated by Snort 1.9.0 and is displayed below by SnortSnarf. The destination port is 80. The flow to server established portion of the signature indicates the three-way handshake has taken place and this is an ACK packet. The signature is looking for the content SEARCH, which is not case sensitive. It expects this value in the 1st 8 bytes of the data (offset=0).

	SnortSnarf alert page
	Source: 202.109.114.237
	SnortSnarf v021024.1

6 such alerts found using input module SnortFileInput, with sources:

- /var/log/snort-elx10/05-13-03/alert
- /var/log/snort-elx10/05-13-03/portscan.log

Earliest: **00:03:01.953533** on 05/13/2003

Latest: **00:05:19.297191** on 05/13/2003

1 different signatures are present for 202.109.114.237 as a source

- 6 instances of [WEB-MISC webdav search access](#)

There are 1 distinct destination IPs in the alerts of the type on this page.

202.109.114.237	Whois lookup at:	ARIN	RIPE	APNIC	Geektools
	DNS lookup at:	Amenesi	TRIUMF	Princeton	
	More lookup links:	Dshield	Sam Spade		

```
[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
```

```
05/13-00:03:01.953533 202.109.114.237:58404 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:4206 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xF84ABFB5 Ack: 0xC8D2A95E Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60862466 0
[Xref => arachnids 474]

[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
05/13-00:03:30.643177 202.109.114.237:58602 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:54025 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xF976680F Ack: 0xC9371E7D Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60865333 0
[Xref => arachnids 474]

[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
05/13-00:03:55.643378 202.109.114.237:58760 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:23736 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xFC6A9880 Ack: 0xC98F43C6 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60867831 0
[Xref => arachnids 474]

[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
05/13-00:04:22.116061 202.109.114.237:59134 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:23201 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xFE0795BD Ack: 0xC9EC73CE Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60870479 0
[Xref => arachnids 474]

[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
05/13-00:04:47.676269 202.109.114.237:59458 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:64652 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xFF8B244C Ack: 0xCA436F17 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60873035 133602390
[Xref => arachnids 474]

[**] [1:1070:5] WEB-MISC webdav search access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
05/13-00:05:19.297191 202.109.114.237:60156 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:7381 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0x107F9A7 Ack: 0xCAB55558 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60876197 0
[Xref => arachnids 474]
```

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Wed May 14 13:02:25 2003

1.3 Probability the Source Address was Spoofed:

The probability that the source address was spoofed is low. The attacker is looking for a response. The TTL of the packets match their source. For example given the alert's TTL of 47 and assuming the attacker is a host that uses 64 for its initial TTL. Now if we do a traceroute back to the host we see it could be 17 hops away, lending support for the idea that it is not spoofed or that the spoofed source is also 17 hops away. Also, the packet is part of an established TCP connection, so it is most likely the packet is not spoofed.

```
C:\>tracert 202.109.114.237

Tracing route to 202.109.114.237 over a maximum of 30 hops

  1  *      *      *      Request timed out.
  2  *      *      *      Request timed out.
  3  171 ms  220 ms  361 ms  12.126.235.233
  4   10 ms   10 ms   10 ms  gbr1-a31s4.cgcil.ip.att.net [12.123.4.78]
  5   20 ms   20 ms   20 ms  tbr2-p013502.cgcil.ip.att.net [12.122.11.49]
  6   20 ms   30 ms   30 ms  tbr2-p012501.sl9mo.ip.att.net [12.122.10.10]
  7   60 ms   70 ms   70 ms  tbr2-p013701.la2ca.ip.att.net [12.122.10.14]
  8   60 ms   70 ms   70 ms  gar1-p370.lsrca.ip.att.net [12.123.199.242]
  9  *      *      *      Request timed out.
 10  411 ms  411 ms  400 ms  202.97.49.66
 11   *      551 ms   *      202.97.51.141
 12  571 ms   *      601 ms  202.97.33.89
 13  591 ms  581 ms  591 ms  202.101.63.233
 14  580 ms  571 ms  581 ms  218.1.1.141
 15  551 ms   *      541 ms  218.1.1.206
 16   *      581 ms   *      218.1.71.210
 17   *      *      *      Request timed out.
 18   *      *      *      Request timed out.
 19   *      *      *      Request timed out.
 20 ^C
C:\>
```

1.4 Description of Attack:

When the Alert was generated (May 13, 2003) Snort was using an out of date rule set (February 9, 2003). There were exploits released in March of the same year. The packet dump is shown below.

```
05/13-00:03:01.953533 202.109.114.237:58404 -> xxx.xxx.xxx.xxx:80
TCP TTL:47 TOS:0x0 ID:4206 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0xF84ABFB5 Ack: 0xC8D2A95E Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 60862466 0
```


53	45	41	52	43	48	20	2F	90	04	CC	04	CC	04	CC	04	SEARCH /
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04
CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04	CC	04

The packet dump output is abbreviated. All six packets **do not** contain same payload. They all look alike the only difference is the repeated portion of the datagram, which contains repetitions of CC 04, CE 04, CA 04, CB 04, CF 04, D1 04. It appears from the packets that the attacker is trying to exploit the WebDAV vulnerability.

The packets are all ACK's, so the three-way handshake has been established. The IP id's and the sequence numbers are all incrementing. The packets are all bound for port 80 and are 1500 bytes long. The packets have time intervals between them of 25 to 30 seconds. All the packets have TCP Options of: NOP NOP TS.

1.5 Attack Mechanism:

Microsoft Security Bulletin MS03-007, in the technical details section states, "WebDAV uses IIS to pass requests to and from Windows 2000. When IIS receives a WebDAV request, it typically processes the request and then acts on it. However, if the request is formed in a particular way, a buffer overrun can result because one of the Windows components called by WebDAV does not correctly check parameters."² This can lead to the attacker being able to execute code or crashing the IIS server.

In this detect there are 6 attempts approximately 25 seconds apart. I believe this is a stimulus because there are services (IIS Web Server 5.0) that communicate with this targeted port. This is a service that has known vulnerabilities as described above. This traffic is not legitimate.

The following text was excerpted from the CERT advisories³.
<http://www.cert.org/advisories/CA-2003-09.html>

"A buffer overflow vulnerability exists in the Win32 API libraries shipped with all versions of Microsoft Windows 2000 and Microsoft Windows NT 4.0. This

vulnerability, which is being actively exploited on WebDAV-enabled IIS 5.0 servers, will allow a remote attacker to execute arbitrary code on unpatched systems. Sites running Microsoft Windows 2000 and Microsoft Windows NT 4.0 should apply a patch or disable WebDAV services as soon as possible.

Microsoft Windows 2000 (and possibly prior versions of Windows) contains a dynamic link library (DLL) named ntdll.dll. This DLL is a core operating system component used to interact with the Windows kernel. A buffer overflow vulnerability exists in ntdll.dll, which is utilized by many different components in the Windows operating system.

The WebDAV ([RFC2518](#)) component of [Microsoft IIS 5.0](#) is an example of one Windows component that uses ntdll.dll. The IIS WebDAV component utilizes ntdll.dll when processing incoming WebDAV requests. By sending a specially crafted WebDAV request to an IIS 5.0 server, an attacker may be able to execute arbitrary code in the Local System security context, essentially giving the attacker complete control of the system.”

1.6 Correlations:

The following is the CVE number for the alert generated by Snort.

From Common Vulnerabilities and Exposures⁴

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109>

Name	CAN-2003-0109 (under review)
Description	Buffer overflow in ntdll.dll, as used by WebDAV on Windows 2000, allows remote attackers to execute arbitrary code via a long request to IIS 5.0.

Arachnids Intrusion Event Database describes this as, an information-gathering attempt, “This event indicates that a remote user has attempted to use the SEARCH directive to retrieve a list of directories on the web server. This may allow an attacker to gain knowledge about the web server that could be useful in an attack.”⁵

The packets received do not quite match, since there is no HTTP/1.1 or Host: or Select directives, as shown in the packet from Arachnids below.

```
01/22-04:02:33.707726 xxx.xxx.xxx.xxx:2543 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x10 ID:60018 IpLen:20 DgmLen:262 DF
***AP*** Seq: 0x6E0098 Ack: 0x4D92C091 Win: 0x7FB8 TcpLen: 20
53 45 41 52 43 48 20 2F 20 48 54 54 50 2F 31 2E  SEARCH / HTTP/1.
31 0D 0A 48 6F 73 74 3A 20 77 68 69 74 65 68 61  1..Host: whiteha
```

```

74 73 2E 63 6F 6D 0D 0A 43 6F 6E 74 65 6E 74 2D  ts.com..Content-
54 79 70 65 3A 20 74 65 78 74 2F 78 6D 6C 0D 0A  Type: text/xml..
43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20  Content-Length:
31 33 33 0D 0A 0D 0A 3C 3F 78 6D 6C 20 76 65 72  133.....
3C 67 3A 73 71 6C 3E 0D 0A 53 65 6C 65 63 74 20  ..Select
22 44 41 56 3A 64 69 73 70 6C 61 79 6E 61 6D 65  "DAV:displayname
22 20 66 72 6F 6D 20 73 63 6F 70 65 28 29 0D 0A  " from scope()..
3C 2F 67 3A 73 71 6C 3E 0D 0A 3C 2F 67 3A 73 65  ....
  
```

The following paragraphs are excerpted from the Snort Intrusion Database⁶. <http://www.snort.org/snort-db/sid.html?id=1070>

Impact	If the target is IIS 5.0, then an attacker may have gotten a complete directory listing from within the web root, which can be useful information for attackers (could be a prelude to a more serious attack). IIS 5.0's WebDAV implementation is also vulnerable to a Denial of Service vulnerability if the search string is too long.
Detailed Information	IIS 5.0 includes an implementation of WebDAV for purposes of web publishing. As shipped, it contains two vulnerabilities that can allow an attacker to get a complete directory listing from the web root and to DoS the web server.
Attack Scenarios	Attacker gets a listing by sending something like: SEARCH / HTTP/1.1 Attacker DoSes the web server using pre-existing tools.

The attack is probably not doing reconnaissance by trying to get a directory listing as described by SID 1070, since the search string does not fit. The attack does however have a long string (the IP header is 20 bytes, the TCP header is 32 bytes leaving 1448 bytes of data), but there is no shellcode. The packets are not fragments since the DF bit is set. Without any apparent shellcode it is unlikely to be a buffer overflow. Looks like the only possibility left is a DOS attempt. SID 1070 mentions the attacker DOSes the web server using pre-existing tools, nothing else is said about these tools.

Joe Stewart⁷ describes several attack tools and signatures, Webdav 1.0.1, Davkit, Webdav 1.1, to name a few. A couple, are said to successfully exploit the vulnerable hosts most of the time. The signatures given do not match this attack.

A Whois using the Geekttools Whois proxy provides the following information.



Final results obtained from whois.apnic.net.

Results:

% [whois.apnic.net node-1]

% How to use this server <http://www.apnic.net/db/>

% Whois data copyright terms <http://www.apnic.net/db/dbcopyright.html>

inetnum: [202.109.0.0](#) - [202.109.127.255](#)

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 20000101

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,PRC

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

Dshield.org returned no hits. Mynetwatchman reported activity from this address for the same time frame as this detect. Most of the activity was reported as a target port of 80 and a probable CodeRed/Nimda. So there is probably no correlation with this activity (from Mynetwatchman).

1.7 Evidence of Active Targeting:

There is quite a bit of evidence of active targeting. The attacker is attacking a specific host with a DOS/exploit that is particular for that application. The attacker probably already has reconnaissance information. It is not a general scan nor is it a wrong number.

1.8 Severity:

Criticality: 4. The host is a web-server, not a critical target. It is hosting an application from which more information could be leveraged, if compromised.

Lethality: 4. The exploit for this has been released. Since this appears to be a DOS, the IIS server did not crash.

System Countermeasures: 3 URL scan is running but no patch applied. If URL scan had not been running this attack could have succeeded.

Network Countermeasures: 2. Firewalled, but traffic allowed to target. There is an IDS running and the firewall is an application level proxy.

$$(4 + 4) - (3 + 2) = 3$$

1.9 Defensive Recommendation:

Since the attacker probably already has reconnaissance information the attacking subnet should be blocked. As new exploits are released, this attacker will not be able to access the target. The Snort rule set should be updated. Implementing SnortSAM might also be considered for these hosts, as there was no advanced warning for these exploits. SnortSAM might also catch new exploits and block connections from new attackers. The patch provided by Microsoft should be applied. There is a discussion of this in [Microsoft Knowledge Base Article 816930](#)⁸. This article describes workarounds that explain:

- How to lock down or disable IIS if your computer does not require it.
- How to disable WebDAV if you do not require it.
- How to use the URL Buffer Size Registry tool.
- How to manually change the MaxClientRequestBuffer registry value if you require WebDAV.

- How to manually create a MaxClientRequestBuffer registry file for a single computer if you require WebDAV.
- How to deploy the MaxClientRequestBuffer registry file through Active Directory by using a Group Policy object.

1.10 Multiple Choice Question

For hosts, actively targeted by an attacker, which of the following is most likely true?

- A. Reconnaissance has not been performed.
- B. Reconnaissance has been performed.
- C. The attack is a general scan of the entire network.
- D. The attack is a wrong number.

Answer B. Reconnaissance has been performed.

2 Network Detect *BAD TRAFFIC* loopback traffic

2.1 Source of Trace:

This trace was captured on my network using Snort 1.9.1 with the ruleset

```
# (C) Copyright 2001,2002, Martin Roesch, Brian Caswell, et al.  
# All rights reserved.  
# $Id: bad-traffic.rules,v 1.18.2.2 2003/02/07 22:04:40 cazz Exp $  
#-----  
# BAD TRAFFIC RULES  
#-----
```

The signature page is displayed below with SnortSnarf. The Snort sensor was sniffing traffic on an internal LAN, when a coworker plugged the management port of a Cisco 4006 supervisor engine into the LAN and booted the device.



1 alerts with this signature using input module SnortFileInput, with sources:

- /var/log/gcia/cisco/07-8-03/alert

Earliest such alert at **09:42:36.148128** on 07/08/2003

Latest such alert at **09:42:36.148128** on 07/08/2003

BAD TRAFFIC loopback traffic	1 sources	1 destinations
Priority: 2	Classification: Potentially Bad Traffic	
[url:rr.sans.org/firewall/egress.php] [sid:528]		
Rules with message "BAD TRAFFIC loopback traffic":		
alert ip any any <> 127.0.0.0/8 any (msg:"BAD TRAFFIC loopback traffic"; classtype:bad-unknown; reference:url,rr.sans.org/firewall/egress.php; sid:528; rev:3;) (from <i>bad-traffic.rules</i>)		

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
127.39.68.188	1	1	1	1

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
51.76.232.210	1	1	1	1

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Thu Jul 10 09:39:53 2003

2.2 Detect Generated by:

The alert was generated by Snort 1.9.1 and is displayed below with SnortSnarf. The device generating the alert was a Cisco Catalyst 4006 with the following software and hardware versions.

```
Console> (enable) sh flash
-#- ED --type-- --crc--- -seek-- nlen -length- -----date/time----- name
 1 .. ffffffff 40cb7582 47b45c 17 4436956 May 27 2003 22:55:03 cat4000.6-4-3.bin
```

```
Console> (enable) sh ver
WS-C4006 Software, Version NmpSW: 6.4(3)
Copyright (c) 1995-2003 by Cisco Systems, Inc.
NMP S/W compiled on Apr 10 2003, 18:23:41
GSP S/W compiled on Apr 10 2003, 16:06:32
```

System Bootstrap Version: 6.1(4)

Hardware Version: 1.2 Model: WS-C4006 Serial #: FOX04477089

Mod	Port	Model	Serial #	Versions
-----	------	-------	----------	----------

1	2	WS-X4013	JAB0419077Y	Hw : 1.2 Gsp: 6.4(3.0) Nmp: 6.4(3)
---	---	----------	-------------	--

The packet has a source address that is in the loopback range. It is IP proto 125, which is TCP Data Flow Control. The packet has a TTL of 235 the TOS is 0x64. It is a fragment with an offset of 8084 bytes (not an 8-bit boundary) and a size of 55568 bytes. The RB, DF and MF bits are set.



SnortSnarf alert page
Source: 127.39.68.188
[SnortSnarf](#) v021024.1

1 such alerts found using input module SnortFileInput, with sources:

- /var/log/gcia/cisco/07-8-03/alert

Earliest: **09:42:36.148128** on 07/08/2003

Latest: **09:42:36.148128** on 07/08/2003

1 different signatures are present for 127.39.68.188 as a source

- 1 instances of [BAD TRAFFIC loopback traffic](#)

There are 1 distinct destination IPs in the alerts of the type on this page.

127.39.68.188	Whois lookup at:	ARIN	RIPE	APNIC	Geektools
	DNS lookup at:	Amenesi	TRIUMF	Princeton	
	More lookup links:	Dshield	Sam Spade		

```
[**] [1:528:3] BAD TRAFFIC loopback traffic [**]  
[Classification: Potentially Bad Traffic] [Priority: 2]  
07/08-09:42:36.148128 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800  
len:0x3C  
127.39.68.188 -> 51.76.232.210 PROTO125 TTL:235 TOS:0x64 ID:8225  
IpLen:20 DgmLen:55588 RB DF MF  
Frag Offset: 0x1F94 Frag Size: 0xB97C
```


[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Thu Jul 10 09:39:53 2003

2.3 Probability the Source Address was Spoofed:

The source address is spoofed as is the source MAC address FE:ED:FA:CE:FE:ED, and the destination MAC address DE:AD:BE:EF:DE:AD.

2.4 Description of Attack:

On bootup of the switch the following packets are sent from the switch's management port. The following packets were caught by Snort's syslog output-plugin on the host chugwater.

```
Jul  8 09:42:36 chugwater snort: [ID 702911 local5.alert] [1:528:3] BAD
TRAFFIC loopback traffic [Classification: Potentially Bad Traffic]
[Priority: 2]: {PROTO125} 127.39.68.188 -> 51.76.232.210

Jul  8 09:42:37 chugwater snort: [ID 702911 local5.alert] [116:1:1]
(snort_decoder) WARNING: Not IPv4 datagram! {IPV6-NONXT}
214.120.143.109 -> 190.143.120.120

Jul  8 09:42:39 chugwater snort: [ID 702911 local5.alert] [116:1:1]
(snort_decoder) WARNING: Not IPv4 datagram! {IPV6-FRAG}
81.15.2.100 -> 173.35.104.78

Jul  8 09:42:40 chugwater snort: [ID 702911 local5.alert] [116:1:1]
(snort_decoder) WARNING: Not IPv4 datagram! {PROTO048}
22.29.68.11 -> 235.45.97.93

Jul  8 09:42:41 chugwater snort: [ID 702911 local5.alert] [116:1:1]
(snort_decoder) WARNING: Not IPv4 datagram! {EGP}
62.8.123.217 -> 28.121.167.39
```

I originally seen this activity in October of 2001, I never saved those packets but I recall that at that time 5 packets were looped back and all were IPv4, so it appears the activity has change. A new switch was recently purchased and I wanted to see if it did the same thing, the result is this detect.

For the original detect a TAC case was opened.

Cisco Case# B921653

Title Strange Packets at Boot Up.

At first they were unable to reproduce it, but finally came back with the response

“These packets are probably generated by the power-on self-test routines which loopback a packet at the management port during testing. This is done to insure the interface is operational and should be harmless.”

There reply seems probable enough but it did not really satisfy me at the time. Particularly the way it was phrased, “probably generated” and “should be harmless”. It also did not seem to be a very good practice to be sending packets onto the LAN to complete a POST test.

On the new switch I only received one detect so I extracted the packets from the binary log with the following command.

```
chugwater:/var/log/gcia/cisco/07-8-03#snort -r snort.log.1057667471 > test
```

```
Log directory = /var/log/snort
```

```
TCPDUMP file reading mode.
```

```
Reading network traffic from "snort.log.1057667471" file.
```

```
snaplen = 1514
```

```
--== Initializing Snort ==--
```

```
Initializing Output Plugins!
```

```
--== Initialization Complete ==--
```

```
-*> Snort! <*-
```

```
Version 1.9.1 (Build 231)
```

```
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

```
[!] WARNING: Not IPv4 datagram! ([ver: 0x0][len: 0x260c])
```

```
[!] WARNING: Not IPv4 datagram! ([ver: 0x6][len: 0xb043])
```

```
[!] WARNING: Not IPv4 datagram! ([ver: 0xa][len: 0x56d5])
```

```
[!] WARNING: Not IPv4 datagram! ([ver: 0x6][len: 0x306a])
```

```
Snort received signal 3, exiting
```

```
chugwater:/var/log/gcia/cisco/07-8-03#
```

According to Northcutt⁹, the IP version field must be validated by the receiving host, and if not valid the datagram will be discarded and no error messages will be sent to the sending host. If a packet arrives at a router with an invalid IP version it should be discarded as well.

2.5 Attack Mechanism:

The packet dumps are shown below.

```
[**] [1:528:3] BAD TRAFFIC loopback traffic [**]  
[Classification: Potentially Bad Traffic] [Priority: 2]
```

```
07/08-09:42:36.148128 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
127.39.68.188 -> 51.76.232.210 PROTO125 TTL:235 TOS:0x64 ID:8225
IpLen:20 DgmLen:55588 RB DF MF
Frag Offset: 0x1F94 Frag Size: 0xB97C

[**] [116:1:1] (snort_decoder) WARNING: Not IPv4 datagram! [**]
07/08-09:42:37.381283 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
214.120.143.109 -> 190.143.120.120 IPV6-NONXT TTL:171 TOS:0x39 ID:13846
IpLen:24 DgmLen:3110 DF

[**] [116:1:1] (snort_decoder) WARNING: Not IPv4 datagram! [**]
07/08-09:42:38.614449 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
81.15.2.100 -> 173.35.104.78 IPV6-FRAG TTL:85 TOS:0x32 ID:28569
IpLen:60 DgmLen:17328

[**] [116:1:1] (snort_decoder) WARNING: Not IPv4 datagram! [**]
07/08-09:42:39.850600 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
22.29.68.11 -> 235.45.97.93 PROTO048 TTL:79 TOS:0x7A ID:36451 IpLen:32
DgmLen:54614 MF

[**] [116:1:1] (snort_decoder) WARNING: Not IPv4 datagram! [**]
07/08-09:42:41.083761 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
62.8.123.217 -> 28.121.167.39 EGP TTL:16 TOS:0x7A ID:13271 IpLen:4
DgmLen:27184 DF

=====
07/08-09:42:36.148128 127.39.68.188 -> 51.76.232.210
PROTO125 TTL:235 TOS:0x64 ID:8225 IpLen:20 DgmLen:55588 RB DF MF
Frag Offset: 0x1F94 Frag Size: 0xB97C
34 4D EA 64 FF 1B 31 31 CD 71 65 02 FA 26 FF 01 4M.d..11.qe..&..
00 13 70 24 DE 2B 38 81 56 31 ..p$.+8.Vl
```

This is quite the assortment of packets. What do we have here, varying TTL's, TOS, Fragments, IPV6, spoofed MAC addresses, spoofed IP addresses.

For the source and destination IP addresses of the packets we have:

2nd Packet

Src OrgName: DoD Network Information Center
Dst OrgName: Internet Assigned Numbers Authority

3rd Packet

Src netname: IS-LINANET-200208
address: 110 Reykjavik
address: Iceland
Dst OrgName: Internet Assigned Numbers Authority

4th Packet

Src OrgName: DoD Network Information Center
Dst OrgName: Internet Assigned Numbers Authority
NetName: MCAST-NET

5th Packet

Src inetnum: [62.8.123.0](#) - [62.8.124.255](#)
netname: HIWAY
descr: Network Interconnects
country: GB
Dst OrgName: DoD Network Information Center
Comment: ARPA DSI JPO

This is an interesting choice of addresses to use for interface testing. This is not an attack but a false positive. Due to the destination MAC address the packets will go to all ports on the switch, but no host or router will pick them up.

2.6 Correlations:

Since the alerts are originating from the management interface of a Cisco switch probably not going to be much in the way of correlations since most people don't use this interface anyway. For the IPv4 packet that generated the original alert a Whois on the destination address using the Geekttools Whois proxy provides the following information.

Final results obtained from [whois.arin.net](#).
Results:
OrgName: Department of Social Security of UK
OrgID: DSSU
Address: Naming and Addressing Authority c/o DITA
Address: Government Buildings - GZI
Address: Moorland Road
Address: Lytham St. Annes, Lancashire FY8 3ZZ
City:
StateProv:
PostalCode:
Country: GB

NetRange: [51.0.0.0](#) - [51.255.255.255](#)
CIDR: [51.0.0.0/8](#)
NetName: ITSANET
NetHandle: NET-51-0-0-0-1
Parent:
NetType: Direct Assignment
Comment:

RegDate: 1991-09-16
Updated: 1999-04-13

For the source IP address of 127.39.68.188 Mynetwatchman reported no incidents for this IP. Dshield also had no info. A Google search on FE:ED:FA:CE:FE:ED produced the following.

<http://www.mcabee.org/lists/snort-users/Oct-01/msg00255.html>

Snort-users] MISC loopback traffic

- *Date:* Tue, 9 Oct 2001 09:36:55 -0500
- *From:* Jim Rauser <jrauser@xxxxxxxxx>
- *To:* snort-users@xxxxxxxxxxxxxxxxxxxxxxxxxx
- *Subject:* [Snort-users] MISC loopback traffic

has anyone seen this before
these are coming off a catalyst 4006 ME1 interface
I get five of these on boot
strange mac addresses

```
[**] [1:528:1] MISC loopback traffic [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
10/08-13:48:36.629261 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
127.39.68.188 <../../../../127/39/68/src127.39.68.188.html> ->
51.76.232.210
<../../../../51/76/232/dest51.76.232.210.html> FIRE TTL:235 TOS:0x64
ID:8225
IpLen:20 DgmLen:55588 RB DF MF
Frag Offset: 0x1F94 Frag Size: 0x1A [**] [1:528:1] MISC loopback
traffic
[**]
[Classification: Potentially Bad Traffic] [Priority: 2]
10/09-08:27:14.759261 FE:ED:FA:CE:FE:ED -> DE:AD:BE:EF:DE:AD type:0x800
len:0x3C
127.39.68.188 <../../../../127/39/68/src127.39.68.188.html> ->
51.76.232.210
<../../../../51/76/232/dest51.76.232.210.html> FIRE TTL:235 TOS:0x64
ID:8225
IpLen:20 DgmLen:55588 RB DF MF
```

jim

Snort-users mailing list
Snort-users@lists.sourceforge.net
Go to this URL to change user options or unsubscribe:
<https://lists.sourceforge.net/lists/listinfo/snort-users>
Snort-users list archive:
<http://www.geocrawler.com/redir-sf.php3?list=snort-users>

2.7 Evidence of Active Targeting:

The attacks are originating from an internal switch. Hard to tell what the intent is. The evidence for active targeting is low. Cisco says they are testing the interface. But is it necessary to inject packets onto the LAN in order to do this? Particularly, with destination addresses that are on the Internet? It makes a person wonder what else it might be doing, or what other coding it has.

2.8 Severity:

Criticality: 3 The device is network infrastructure.

Lethality: 0 This is benign a false positive.

System Countermeasures: 1 The management interface is not used.

Network Countermeasures: 3 Firewall in place. There is an IDS. Egress filters on internal routers.

$$(3 + 0) - (1 + 3) = -1$$

2.9 Defensive Recommendation:

This is a false positive, however a person may want to still use egress filters. Don't use the management interface on the Catalyst 4006.

2.10 Multiple Choice Question

What happens to an Ethernet frame with an incorrect destination MAC address (not the default router) when it is put on the LAN and is not destined for this subnet?

- A. The frame is broadcast to all devices on the switch, the default router processes it by stripping off the frame, encapsulating it for the proper media and routing it to the next hop router.
- B. The switch drops the packet since it does not recognize the MAC address.
- C. The frame is broadcast to all devices on the switch, but none of them process the frame, since they are not the destination.
- D. The switch ARP's for the destination IP and rewrites the layer 2 header with the received MAC address of the default router.

Answer C.

3 Network Detect (spp_portscan2) Portscan detected

3.1 Source of Trace:

The trace was taken from the raw TCPdump logs at <http://www.incidents.org/logs/RAW>. The file used (2002.9.11) was created using an instance of Snort running in binary mode. All of the non-local addresses have been munged. The SnortSnarf signature page is shown below. The snort.conf file was modified to turn on the Snort preprocessor for port scanning. From the layer 2 addresses it appears that the Snort sensor is between two Cisco routers.

	SnortSnarf signature page (spp_portscan2) Portscan detected SnortSnarf v021024.1
---	--

1 alerts with this signature using input module SnortFileInput, with sources:

- /var/log/gcia/syn-ack/alert

Earliest such alert at **11:40:16.496507** on 10/11/2002

Latest such alert at **11:40:16.496507** on 10/11/2002

(spp_portscan2) Portscan detected	1 sources	1 destinations
Priority: N/A	Classification: N/A	

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
129.186.23.70	1	1	1	1

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
32.245.196.182	1	1	1	1

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Fri Jun 6 13:49:48 2003

	SnortSnarf signature page spp_portscan2: TCP ***A**S* scan SnortSnarf v021024.1
---	---

2 alerts with this signature using input module SnortFileInput, with sources:

- /var/log/gcia/syn-ack/alert
- /var/log/gcia/syn-ack/scan.log

Earliest such alert at **11:40:16.496507** on 10/11/2002

Latest such alert at **11:40:35.166507** on 10/11/2002

spp_portscan2: TCP ***A**S* scan	1 sources	2 destinations
Priority: N/A	Classification: N/A	

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
129.186.23.70	2	3	2	2

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
32.245.114.180	1	1	1	1
32.245.196.182	1	2	1	1

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Tue Jun 17 20:52:14 2003

3.2 Detect Generated by:

The file was extracted using the following Snort 1.9.1 command:
snort -r 2002.9.11 -c snort.conf -l /var/log/gcia/2002.9.11 -de. The SnortSnarf alert page is shown below.

	SnortSnarf alert page Source: 129.186.23.70 SnortSnarf v021024.1
---	--

3 such alerts found using input module SnortFileInput, with sources:

- /var/log/gcia/syn-ack/alert
- /var/log/gcia/syn-ack/scan.log

Earliest: **11:40:16.496507** on 10/11/2002

Latest: **11:40:35.166507** on 10/11/2002

2 different signatures are present for 129.186.23.70 as a source

- 1 instances of [\(spp_portscan2\) Portscan detected](#)
- 2 instances of [spp_portscan2: TCP ***A**S*.scan](#)

There are 2 distinct destination IPs in the alerts of the type on this page.

129.186.23.70	Whois lookup at:	ARIN	RIPE	APNIC	Geektools
	DNS lookup at:	Amenesi	TRIUMF	Princeton	
	More lookup links:	Dshield	Sam Spade		

```
[**] [117:1:1] (spp_portscan2) Portscan detected from 129.186.23.70: 6
targets 6 ports in 2582 seconds [**]
10/11-11:40:16.496507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x3C
129.186.23.70:6000 -> 32.245.196.182:3072 TCP TTL:45 TOS:0x0 ID:0
IpLen:20 DgmLen:44 DF
***A**S* Seq: 0x11D2EAE6 Ack: 0x895C0042 Win: 0x16D0 TcpLen: 24
TCP Options (1) => MSS: 1460
```

```
10/11-11:40:16.496507 TCP 129.186.23.70:6000 -> 32.245.196.182:3072
tgts: 6 ports: 6 flags: ***A**S* event_id: 0
```

```
10/11-11:40:35.166507 TCP 129.186.23.70:6000 -> 32.245.114.180:1024
tgts: 7 ports: 7 flags: ***A**S* event_id: 2
```

[SnortSnarf](#) brought to you courtesy of [Silicon Defense](#)

Authors: [Jim Hoagland](#) and [Stuart Staniford](#)

See also the [Snort Page](#) by Marty Roesch

Page generated at Tue Jun 17 20:52:14 2003

I was also able to extract the following packets using the command:

```
snort -r 2002.9.11 -c snort-1.9.1/rules/snort.conf -l /var/log/gcia/syn-ack -
de tcp[13]=18.
```

This was done to get all the packets with both the SYN and ACK bits set

[illegible]

There is one packet, followed by another one in twenty minutes. The next 5 packets were received within a 60 second timeframe.

It appears that an attacker is sending SYN packets to 129.186.23.70, with spoofed sources, of 32.245.242.221, 32.245.104.220, 32.245.62.155, 32.245.62.155, 32.245.104.125, etc. (our network space). We are receiving the SYN-ACK's, from 129.186.23.70 in response to this.

3.5 Attack Mechanism:

Even without knowing what services are running on the target hosts we can say this is response, and it is not legitimate traffic. From Northcutt¹¹, “a SYN packet sent to a open port results in a SYN-ACK packet being sent to the spoofed IP address. If it exists, the spoofed IP address responds to this unexpected SYN-ACK with a RST back to the victim machine.” Even though we don't have the RST packet, this is most likely what happened.

One thing that points to this is as follows, if it was legitimate traffic, it is very unlikely that 5 of our hosts sent a SYN packet to the same server on the same source port and are awaiting a SYN-ACK all within a time window of 60 seconds. Another odd thing is, would all the IP ID's be zero if the traffic were legitimate? The range for valid IP ID's is 1-65535. For each datagram sent a host sends, it must generate a unique ID number that is typically incremented by one for each new datagram¹⁰. Zero is not a typical value, indicating some kind of packet crafting is being done. Of course the attacker could be doing recon by just sending us SYN-ACKS since 1024 is a known Trojan port, but I'll stick with the third party effect given the limited evidence.

This is probably a reflection from a DOS/recon attack against 129.186.23.70, using our hosts as spoofed addresses. So it appears to the victim that the attack is originating from us. It seems that the attacker could have knowledge of hosts on our subnets as witnessed by the large gaps in between the hosts. This is benign or possibly reconnaissance. No attack tools with this signature were found.

3.6 Correlations:

A Google search produced the following correlation from GIAC:

(Curt Wilson)¹²

INTERNET SUSPICIOUS ACTIVITY - Dec 26 - Jan 31, 2000

Detect 1: Scanning attempt from what appears to be an IRC server (astro.ga.us.dal.net,

port 6667) to TCP port 1024 for every IP in our external range; which alternates with destination port 3072. An IRC server does exist on 64.154.61.232. This activity continued at various intervals throughout the day. Research through securityfocus.com's incident list has shown that there is a coordinated attack of IRC servers taking place with spoofed IP addresses or decoys. Attacker has picked our IP addresses to appear as the attacking systems. This does not damage our systems, but makes it appear to the receiving end that we are attacking them. There is no way to stop this behavior short of applying egress filtering to the attackers network, and since the attacker can't be tracked from our logs, there is nothing we can do. Risk: low

```
Dec 26 11:25:24 [firewall.ip.address] %PIX-7-106011: Deny inbound (No
xlate) tcp src outside:64.154.61.232/6667 dst outside:cidr.net.addr.107/1024
Dec 26 11:34:29 [firewall.ip.address] %PIX-7-106011: Deny inbound (No
xlate) tcp src outside:64.154.61.232/6667 dst outside:cidr.net.addr.106/3072
Dec 26 11:34:29 [firewall.ip.address] %PIX-7-106011: Deny inbound (No
xlate) tcp src outside:64.154.61.232/6667 dst outside:cidr.net.addr.106/3072
Dec 26 13:34:55 [firewall.ip.address] %PIX-7-106011: Deny inbound (No
xlate) tcp src outside:64.154.61.232/6667 dst outside:cidr.net.addr.105/1024
Dec 26 13:34:55 [firewall.ip.address] %PIX-7-106011: Deny inbound (No
xlate) tcp src outside:64.154.61.232/6667 dst outside:cidr.net.addr.105/1024
```

A Whois using the Geekttools Whois proxy provides the following information about the source.

Final results obtained from whois.arin.net.

Results:

OrgName: Iowa State University
OrgID: IAST
Address: Academic Information Technologies
Address: 291 Durham Hall
City: Ames
StateProv: IA
PostalCode: 50011
Country: US

NetRange: 129.186.0.0 - 129.186.255.255

CIDR: 129.186.0.0/16

NetName: CYCLONENET

NetHandle: NET-129-186-0-0-1

Parent: NET-129-0-0-0-0

NetType: Direct Assignment

NameServer: NS-3.IASTATE.EDU

NameServer: NS-2.IASTATE.EDU

```
NameServer: NS-1.IASTATE.EDU
NameServer: SCSDS.AMESLAB.GOV
Comment:
RegDate: 1988-03-17
Updated: 1998-04-10

TechHandle: TC42-ARIN
TechName: Contact, Technical
TechPhone: +1-515-294-2256
TechEmail: tech-contact@iastate.edu

OrgAbuseHandle: ABUSE110-ARIN
OrgAbuseName: Abuse Contact
OrgAbusePhone: +1-515-294-2256
OrgAbuseEmail: abuse@iastate.edu

OrgTechHandle: TC42-ARIN
OrgTechName: Contact, Technical
OrgTechPhone: +1-515-294-2256
OrgTechEmail: tech-contact@iastate.edu

# ARIN WHOIS database, last updated 2003-06-08 21:05
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

It seems reasonable that a university would have a host with ports 1024 and 3072 open. It also seems reasonable that this host would be the victim of an attack. That and with the above correlation it would appear that an attack is underway against 129.186.23.70 using our spoofed addresses.

3.7 Evidence of Active Targeting:

The evidence for active targeting is low. Even though we don't know the services that are running. We have determined that this is not traffic destined for these hosts.

3.8 Severity:

Criticality: 0 We don't know what services the hosts are running. But we have determined the traffic is not destined for them.

Lethality: 1 This is probably benign, at the worst reconnaissance.

System Countermeasures: 2 Little known here. Is the service running?

Network Countermeasures: 2 Is there a firewall? Are these ports blocked?

$$(0 + 1) - (2 + 2) = -3$$

3.9 Defensive Recommendation:

If a firewall is in place the ports should be closed. Blocking the source IP is probably not necessary since this is most likely a third party effect. Implementing a TCPdump audit trail would give additional information in the future, such as the RST packets and/or any ICMP generated. Participating in a distributed intrusion detection such as Dshield would help stop this activity sooner rather than later.

Checking the destination hosts to see what services are running would also be helpful.

3.10 Multiple Choice Question:

For a given three-way handshake and that has a SYN packet with a different TTL than the RST packet, what is the most likely situation that caused this?

- a. The port is open on the destination host.
- b. The network path changed.
- c. The address is spoofed.
- d. The port is closed on the destination host.

Answer C. The address is spoofed.

Three Questions from Post

Only received one reply from my post on 6/16/03. <http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00175.html>. After, the post I redid my Source of Trace and Detect Generated sections. When SnortSnarf generated the html I had it include the scans.log, this file was missed inadvertently. This did not affect the analysis just displayed the data a little differently.

-----Original Message-----

From: Oliver Viitamaki [mailto:ov@mdsi.bc.ca]
Sent: Monday, June 16, 2003 10:33 AM
To: Jim Rauser; intrusions@incidents.org
Subject: jrauser@bepec.com - Email found in subject - Re: LOGS: GIAC
GCIA
Version 3.3 Practical Detect(s)jrauser

At 09:13 AM 6/16/2003 -0500, you wrote:
>Network Detect (spp_portscan2) Portscan detected

```
[**] [117:1:1] (spp_portscan2) Portscan detected from 129.186.23.70: 6
targets 6 ports in 2582 seconds [**]
10/11-11:40:16.496507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x3C
129.186.23.70:6000 -> 32.245.196.182:3072 TCP TTL:45 TOS:0x0 ID:0
IpLen:20
DgmLen:44 DF
***A**S* Seq: 0x11D2EAE6 Ack: 0x895C0042 Win: 0x16D0 TcpLen: 24
TCP Options (1) => MSS: 1460
```

1.4 Description of Attack:

This is third party effect picked up initially by the port scan detector.

The attack is a SYN-ACK to ports 1024 and 3072. The SYN-ACK flags, in normal operation, indicate that this is a reply. Each packet has an IP Id of

zero and one TCP option of: MSS 1460. The time interval is interesting. There is one packet, followed by another one in twenty minutes. The next 5

packets were received within a 60 second timeframe.

It appears that an attacker is sending SYN packets to 129.186.23.70, with

spoofed sources, of 32.245.242.221, 32.245.104.220, 32.245.62.155, 32.245.62.155, 32.245.104.125, etc. (our network space). We are receiving

the SYN-ACK's, from 129.186.23.70 in response to this.

Question. What additional evidence do you have available that you may not have drawn out or included to indicate that this traffic is not "normal" XWindows? How is this not a false positive?

Over

I believe I cover this in the Attack Mechanism section in the original analysis with the following paragraph.

One thing that points to this is as follows, if it was legitimate traffic, it is very unlikely that 5 of our hosts sent a SYN packet to the same server on the same source port and are awaiting a SYN-ACK all within a time window of 60 seconds. Another odd thing is, would all the IP id's be zero if the traffic were legitimate? Of course the attacker could be doing recon on our hosts, by just sending us SYN-ACKS since 1024 is a known Trojan port, but I'll stick with the third party effect given the limited evidence.

This alert could possibly also be a false positive, but since we have very little information about the destination hosts, third party effect still seems likely.

© SANS Institute 2004, Author retains full rights.

References

- ¹ Hoagland, Jim and Staniford Stuart. "SnortSnarf."
URL: <http://www.silicondefense.com/software/snortsnarf/>
- ² Microsoft Corporation. "Microsoft Security Bulletin MS03-007"
URL: <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms03-007.asp>
- ³ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center."
URL: <http://www.cert.org/advisories/CA-2003-09.html>
- ⁴ The MITRE Corporation. "Common Vulnerabilities and Exposures (CVE)"
URL: <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109>
- ⁵ Vision, Max. "Arachnids Intrusion Event Database."
URL: <http://www.whitehats.com/info/IDS474>
- ⁶ Roesch, Martin. "The Snort Signature Database."
URL: <http://www.snort.org/snort-db/sid.html?id=1070>
- ⁷ Stewart, Joe. Webdav Exploits Exposed.
URL: <http://www.lurhq.com/webdav.html>
- ⁸ Microsoft Corporation. "MS03-007: How to Work Around the Vulnerability That Is Discussed in Microsoft Knowledge Base Article 815021."
URL: <http://support.microsoft.com/default.aspx?scid=kb;en-us;816930>
- ^{9 10} Northcutt, Stephen. Track 3 Intrusion Detection In-Depth. 2003 SANS Institute
- ¹¹ Northcutt, Stephen et. al. Intrusion Signature Analysis. 2001 New Riders Publishing.
- ¹² Wilson, Curt. Posts to GIAC. Report Date: January 9, 2001
URL: <http://www.sans.org/y2k/010901-1300.htm>

Assignment 3 Analyze This

Executive Summary

Introduction

Analyze this is a review of 5 days of Snort¹ logs from a major university as a part of a Security Audit. Snort is a packet logger and intrusion detection system. The way Snort works is; it captures packets off a network and compares the packets to known signatures of various suspect or malicious activity. When it finds a match an alert or detect is generated that includes the type, source, destination and other identifying characteristics of the underlying transport protocol TCP/IP.

Attackers typically take advantages of weaknesses in TCP/IP to accomplish several things: Reconnaissance, identifying live hosts and Operating System fingerprinting, which is identifying the host OS. This is invaluable information to an attacker as it allows them to actively target or pinpoint known vulnerabilities in the OS. For the attacker it is either about gaining access or preventing others from access (Denial of Service), either using application/OS attacks or network attacks. Once access is gained the attacker attempts to maintain access through: Trojans, Backdoors and Rootkits.

An Intrusion Detection System such as Snort aids in thwarting these events. It does not stop them but alerts to the presence of them. The system is not perfect and requires an experienced analyst to tune and review the alerts generated. One of the problems with Snort and any other IDS for that matter, is false positives. These are alerts that are not suspect or malicious they are normal traffic that just happens to match the signature. This is where the experience analyst comes in. It is their job to recognize these and/or tune Snort so it does not produce these, as they can become excessive. This can be a problem as it is distracting from the genuine alerts.

The sensors appear to be running a version of Snort about 1.9.0 or before. There are probably quite a few of them. Some are generating a high number of false positives and need to be tuned. Others are being used in packet logging mode and need to be removed from the IDS. They could be set up as part of a TCPdump audit trail.

Current state of the network

At the start of the analysis little is known about the network from which the logs came from. But working through the analysis of the logs provides clues about the topology of the network. It is a very large network with 100+ subnets and 1000+ hosts. To provide services to so many hosts quite a large amount of networking gear is required. It is a very open network, in that, here

does not appear to be many if any filtering devices or the filters are very relaxed. The usual infrastructure is present DNS, Mail, Web Services, FTP, Dial-up, and Internet Access.

In the course of the analysis logs from 7/5 to 7/9 were analyzed. The alerts logs contained 61 unique alerts with a total of 27614 alerts. There were 18679 source addresses and 4337 destination addresses. A timeline was done over the five days. The average number of alerts per hour was between 1000 to 3000. With peaks of 10,000 to 13000 per hour on 7/7.

Evidence of host compromise was found in abundance. There appears to be a Code Red infection. Trojan and/or remote control activity was also found. Reconnaissance is being performed and is ongoing. Active targeting of hosts for specific known vulnerabilities is also being performed. There is some evidence that could point to poor network design; from the presence of traffic not originating or bound to these networks.

It is evident that some type of packet filtering needs to be done, to block access to these LAN's from external connections. It also seems a review of the present policies and procedures concerning access and known vulnerabilities needs to be initiated. It appears that systems with known vulnerabilities are not being patched in a timely manner. Also the level of access to internal hosts (IRC activity for example) needs to be reviewed and restricted. These two actions alone would put a stop to a majority of this activity from external sources.

List of Files Analyzed

The Out of Spec files were off by a day.

Alerts	Scans	OOS
Alert.030705	Scans.030705	OOS_Report_2003_07_06_23454.txt
Alert.030706	Scans.030706	OOS_Report_2003_07_07_25549.txt
Alert.030707	Scans.030707	OOS_Report_2003_07_08_5584.txt
Alert.030708	Scans.030708	OOS_Report_2003_07_09_2126.txt
Alert.030709	Scans.030709	OOS_Report_2003_07_10_4402.txt

Analysis Process

To analyze the log files I chose to use a database. To prepare for this I installed MySQL, an Open Source RBMS. I modified logsnorter-0.2² to include a subroutine for processing the logs from <http://www.incidents.org/logs>. Logsnorter parses through log files and generates MySQL statements to insert the logs into the database. An Apache web server was installed along with PHP (Web scripting language) the implementation language of ACID. ADODB the PHP

database abstraction library and PHPlot were installed. Various graphing libraries were also installed and finally ACID³.

I also installed Webmin, to make it easier to manipulate the MySQL databases, such as adding db's, host and user permissions and executing SQL statements in order to import the schema. I created a database for each of the alerts, scans, and OOS.

The OOS and alert files all had their addresses obfuscated. Before further processing, I replaced the MY.NET using the following script.

```
cat alert.030705 | ./replace > alert.030705

#!/usr/bin/perl
while (my $line = <STDIN>) {
    if ($line =~ s/MY.NET/my.net/g) {
        print $line;
    } else {
        print $line;
    }
}
```

The databases were then populated using the following sets of commands one for each file:

```
tensleep:[/var/log/gcia]#cat logs/alert/alert.030705 | ./logsnorter-0.2 -t -u shotgun
-d alert -p password -s localhost -L 16
```

```
tensleep:[/var/log/gcia]#cat logs/oos/OOS_Report_2003_07_06_23454.txtr |
./logsnorter-0.2 -t -u shotgun -d oos -p password -s localhost -L 16
```

```
tensleep:[/var/log/gcia]#cat logs/scans/scans.030705 | ./logsnorter-0.2 -t -u
shotgun -d scans -p password -s localhost -L 16
```

The -t option uses the logs timestamps, the -u is the mysql user, -d the database to insert into, -p the mysql password, -s the host the database resides on, and -L the type of logfile to process.

The alerts files have little means for distinguishing udp from tcp traffic so udp was tagged as tcp in the SQL database.

Each of the unique alerts was analyzed looking for false positives, compromised hosts, correlations and any defensive recommendations.

List of Detects - Description, Analysis, Correlations, Defensive Recommendations

Analysis Console for Intrusion Databases

Queried on : Thu July 24, 2003 10:14:00

Database: snort@localhost (schema version: 106)

Time window: [2003-07-05 00:12:47] - [2003-07-09 23:46:55]

Sensors: 1 Unique Alerts: 61 (1 categories) Total Number of Alerts: 276141 <ul style="list-style-type: none">Source IP addresses: 18679Dest. IP addresses: 4337Unique IP links 62365Source Ports: 46532<ul style="list-style-type: none">TCP (46532) UDP (0)est. Ports: 1258<ul style="list-style-type: none">TCP (1258) UDP (0)	Traffic Profile by Protocol TCP (100%) <div></div> UDP (0%) <div></div> ICMP (< 1%) <div></div> Portscan Traffic (0%) <div></div>
---	--

Note: Most of the descriptions are the work of others and are properly referenced.

Signature	Total #	Src. Addr.	Dest. Addr.
CS WEBSERVER - external web traffic	96180 (35%)	15438	6
Description: A custom rule to detect traffic on port 80. Analysis: The majority of detects from this rule will probably be false positives, unless external traffic is not allowed to the server. This does not appear to be the case. A better place for this would be part of a TCPdump audit trail. It will be difficult to weed out normal traffic from malicious using this rule alone. Well let's see what we can do with these "alerts" anyway. There are 42862 unique source ports. Ports 137, 213 and 721 stick out. The six destination hosts are:			

≤ Dest IP address ≥	≤ Total # ≥	≤ Src. Addr. ≥
my.net.30.4	1	1
my.net.60.14	1	1
my.net.100.165	96169	15437
my.net.132.57	1	1
my.net.133.206	1	1
255.255.255.255	7	7

The broadcast address is interesting:

≤ Signature ≥	≤ Timestamp ≥	≤ Source Address ≥	≤ Dest. Address ≥	≤ Layer 4 Proto ≥
CS WEBSERVER - external web traffic	2003-07-08 08:40:17	66.196.72.106	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-09 15:47:18	65.128.184.16	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-09 23:34:46	152.163.252.65	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-09 23:38:40	66.196.72.67	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-07 04:43:04	132.185.240.13	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-06 15:16:29	64.68.82.50	255.255.255.255	ICMP
CS WEBSERVER - external web traffic	2003-07-06 16:41:44	66.196.72.73	255.255.255.255	ICMP

This is interesting that this rule picks this up. Broadcast traffic from external hosts can indicate a problem with the network design. This needs to be corrected; the routers should not be forwarding broadcast packets. Or these could be spoofed sources originating from this network in this case egress filters should drop them.

There is just not enough information here to make a determination as to whether or not the hosts have been compromised.

Correlations:

Defensive Recommendations: This rule appears to be used for logging connections. This rule will detect on all normal traffic to the web server, greatly increasing the number of detects which are false positives. This in effect, detracts from the effectiveness of the IDS by producing a large amount of false positives that an analyst must wade through. This rule is producing almost a third of the alerts. The host web server logs all this activity, so this is redundant. Snort is a great packet logger, but the output should probably not be included with the other IDS detects. Use the snort rule set for detecting malicious traffic and setup another sensor for a TCPdump audit trail for capturing all packets in a connection. To be used to reconstruct what happened in the event of an actual attack. There are other rules that this applies to also.

[spp_http_decode: IIS Unicode attack detected](#)⁴

[62987](#)
(23%)

[535](#)

[838](#)

Description: Use the above hyperlink.

Analysis:

Wow, this is a lot of alerts. Lets just focus on the ones to internal hosts. The external destinations are probably false positives generated by surfing activity. So search on this signature where the destination is my.net.0.0/16 and the source is not. This narrows it down to 2748 alerts, to 127 destinations. Still quite a few to analyze. If we had some payload we could search on that also. So without any content or knowledge of the hosts it is hard to say anymore. Need to match up the lists of hosts with the known vulnerability to these 127 destinations and go from there.

≤ Dest IP address ≥	≤ Total # ≥	≤ Src. Addr. ≥
my.net.5.15	87	2
my.net.5.20	84	1
my.net.5.44	83	1
my.net.5.45	74	1
my.net.5.46	111	8
my.net.5.55	82	1
my.net.5.67	84	2
my.net.5.92	83	1
my.net.5.95	84	1
my.net.6.7	52	10
my.net.6.14	1	1
my.net.7.98	2	1
my.net.7.103	7	1
my.net.9.5	2	1
my.net.17.2	7	1
my.net.17.4	3	1
my.net.17.45	1	1
my.net.18.45	1	1
my.net.18.47	2	1
my.net.21.2	7	1
my.net.21.4	7	1
my.net.21.6	5	1
my.net.21.7	10	2
my.net.21.15	1	1
my.net.21.16	6	1
my.net.21.23	1	1
my.net.21.25	7	1
my.net.21.26	4	1
my.net.21.27	23	5
my.net.21.29	4	1
my.net.21.43	35	7

my.net.21.44	3	1
my.net.21.47	2	1
my.net.21.48	6	1
my.net.21.51	25	4
my.net.21.54	1	1
my.net.21.60	4	3
my.net.21.64	6	1
my.net.21.71	6	1
my.net.21.72	7	1
my.net.21.81	6	1
my.net.21.101	6	1
my.net.22.36	2	1
my.net.22.82	5	1
my.net.24.34	55	11
my.net.24.37	7	1
my.net.24.44	68	29
my.net.24.46	7	1
my.net.29.8	74	1
my.net.29.12	82	1
my.net.29.18	83	1
my.net.29.66	8	1
my.net.31.7	7	1
my.net.32.133	7	1
my.net.32.139	7	1
my.net.32.141	1	1
my.net.32.143	3	1
my.net.32.144	4	1
my.net.32.149	3	1
my.net.32.153	1	1
my.net.32.155	5	2
my.net.32.157	7	1
my.net.32.160	8	1
my.net.32.166	6	1
my.net.32.169	6	1
my.net.32.170	7	1
my.net.32.175	1	1
my.net.32.183	4	1
my.net.32.184	5	1
my.net.60.14	26	13
my.net.60.17	5	1
my.net.62.17	3	1
my.net.65.20	4	1
my.net.69.145	41	2
my.net.69.243	7	1
my.net.70.118	5	1
my.net.70.135	8	2

my.net.80.144	3	1
my.net.80.161	5	1
my.net.86.19	87	7
my.net.86.26	6	1
my.net.98.99	10	1
my.net.99.37	7	1
my.net.100.69	1	1
my.net.100.165	82	15
my.net.108.52	17	3
my.net.109.9	2	2
my.net.109.73	6	1
my.net.110.47	7	1
my.net.110.80	2	1
my.net.110.224	91	1
my.net.111.15	14	2
my.net.111.21	84	1
my.net.111.42	4	1
my.net.111.140	140	23
my.net.111.184	5	1
my.net.112.216	82	1
my.net.113.208	65	1
my.net.114.31	1	1
my.net.115.12	1	1
my.net.115.44	6	1
my.net.115.55	6	1
my.net.115.130	5	1
my.net.135.148	1	1
my.net.137.18	80	1
my.net.141.31	3	1
my.net.150.83	1	1
my.net.150.228	74	2
my.net.151.16	7	1
my.net.153.105	1	1
my.net.153.219	1	1
my.net.153.221	1	1
my.net.154.30	7	1
my.net.154.31	5	1
my.net.157.11	4	1
my.net.157.24	6	1
my.net.158.254	5	1
my.net.162.37	1	1
my.net.163.142	2	1
my.net.177.65	6	1
my.net.180.25	3	1
my.net.180.28	6	1
my.net.180.29	6	1

my.net.184.25	84	1
my.net.184.47	84	1
my.net.184.251	7	1
my.net.198.214	10	1

Correlations:

Defensive Recommendations: Check hosts that have the vulnerability with the list of 127 destinations. Any matches should be checked for signs of compromise. If compromised rebuild and patch. User access should be restricted to an assigned web root directory and subdirectories when interacting with a web server. Attackers who attempt to perform directory traversals outside the web root should be denied access. Use a BPF filter to ignore outbound web traffic: snort <options> not port 80 and not net my.net (or whatever your network is), or tell the snort preprocessor not to process this alert. preprocessor http_decode: 80 -unicode -cginull

[SMB Name Wildcard](#) ⁵

[46452](#)
(17%)

[675](#)

[1759](#)

Description: Use above hyperlink.

Analysis:

This type of query, when originating from an external network, is usually a pre-attack probe to gather netbios name table information such as workstation name, domain, and a list of currently logged in users. There are a lot of external connections here, some of the top ones are:

	Total	Dest. Addr.
61.219.26.178	121	121
61.220.40.43	122	122
61.220.40.44	175	175
24.117.55.43	871	1
61.221.251.65	216	216
64.208.118.94	170	170
61.163.141.229	234	234
61.183.33.100	147	147
61.230.53.152	76	76
61.230.168.235	219	219
62.29.114.120	157	157
61.231.52.231	174	174
61.231.97.217	58	58
64.38.91.92	241	241
65.43.167.180	166	166

64.76.2.207	182	182
62.7.132.222	84	84
62.29.64.7	123	123
62.29.76.209	111	111

The 12.0.0.0/8 network had a lot of addresses with 2 or 3 detects.

Correlations:

Defensive Recommendations: Block external connections.

my.net.30.4 activity	22107 (8%)	413	3
-----------------------------	-------------------------------	---------------------	-------------------

Description: A custom rule that detected traffic to ports 21, 22, 80, 82, 443, 524, 3372, 3389, 3410, 4899, 45000, and 51433.

Analysis:

From the unique TCP destination ports table, we can see that most of the traffic is to ports 80 and port 524. Pot 524 is ncp. Probably false positives.

≤ Port ≥	≤ Occurrences ≥	≤ Source IP ≥	≤ Dest. IP ≥	≤ First ≥	≤ Last ≥
21 / tcp	1	1	1	2003-07-09 00:57:19	2003-07-09 00:57:19
22 / tcp	2	2	1	2003-07-06 00:59:42	2003-07-06 02:09:29
80 / tcp	16922	390	1	2003-07-05 00:14:58	2003-07-09 23:37:19
82 / tcp	2	2	1	2003-07-08 05:16:57	2003-07-08 05:16:59
443 / tcp	2	1	1	2003-07-07 12:35:49	2003-07-07 12:35:50
524 / tcp	682	12	1	2003-07-06 03:59:20	2003-07-09 22:23:31
3372 / tcp	1	1	1	2003-07-08 21:52:38	2003-07-08 21:52:38
3389 / tcp	2	1	1	2003-07-08 21:55:56	2003-07-08 21:55:57
3410 / tcp	1	1	1	2003-07-09 00:05:46	2003-07-09 00:05:46
4899 / tcp	1	1	1	2003-07-09 16:18:27	2003-07-09 16:18:27
34287 / tcp	1	1	1	2003-07-06 14:44:04	2003-07-06 14:44:04
45000 / tcp	3	1	1	2003-07-09 00:48:54	2003-07-09 00:48:55
51443 / tcp	4486	3	1	2003-07-05 23:23:44	2003-07-08 21:16:55

The other activity on the high ports needs more investigation.

195.223.97.170 :1497	my.net.30.4 :3372
217.120.249.241 :1965	my.net.30.4 :4899
65.165.88.1 :40137	my.net.30.4 :51443

All this external activity is very suspicious. Check host for compromise.

Correlations:

Defensive Recommendations: Check hosts for compromise. Rebuild confirmed compromised hosts. Turn off unused services. Block external connections. This rule appears to be used for logging connections. Snort is a great packet logger, but the output should probably not be included with the other IDS detects.

Queso fingerprint	6474 (2%)	284	78																		
Description: Queso is a simple program to find out what OS your host is using. By sending obscure and/or bogus TCP flags combinations such as 1 and 2, with a SYN, the attacker attempts to identify the TCP/IP stack from the responses.																					
Analysis: The attackers are doing reconnaissance. Scanned hosts may in the future be the recipients of attacks from the scanners.																					
Correlations:																					
Defensive Recommendations: Block external connections.																					
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC⁶	5864 (2%)	2	2																		
Description: Use above hyperlink.																					
Analysis: my.net.198.221 --> 205.188.149.12 undernet.irc.aol.com my.net.74.216 --> 212.161.35.251 beethoven.kewl.org All but one of the alerts comes from my.net.198.221. Using well-known Trojan ports. Possible compromise.																					
Correlations:																					
Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised hosts. Block access to external connections. Set rules for each computer attacked to the network, that each must log into their machine with a password and not a default Administrator account. Also, disable file sharing on all machines.																					
High port 65535 tcp - possible Red Worm – traffic⁷	5387 (2%)	97	151																		
Description: Use above hyperlink. A custom rule that appears to look for a destination of port 65535, as a result there may be quite a few false positives from this rule. Don't know if this is a protocol rule or also includes content.																					
Analysis: We have alerts with both source and destination from this network. Lets look at the destinations first. We have 2315 alerts to 91 destinations. Some of this appears to be mail traffic which could be false positives. Lets filter that out also. So here are the destinations reporting this alert. Possible compromised hosts.																					
<table><tr><td>≤ Dest IP address ≥</td><td>≤ Total# ≥</td><td>≤ Src. Addr. ≥</td></tr><tr><td>my.net.4.80</td><td>1</td><td>1</td></tr><tr><td>my.net.5.20</td><td>5</td><td>1</td></tr><tr><td>my.net.5.239</td><td>1</td><td>1</td></tr><tr><td>my.net.6.63</td><td>11</td><td>1</td></tr><tr><td>my.net.9.31</td><td>1</td><td>1</td></tr></table>				≤ Dest IP address ≥	≤ Total# ≥	≤ Src. Addr. ≥	my.net.4.80	1	1	my.net.5.20	5	1	my.net.5.239	1	1	my.net.6.63	11	1	my.net.9.31	1	1
≤ Dest IP address ≥	≤ Total# ≥	≤ Src. Addr. ≥																			
my.net.4.80	1	1																			
my.net.5.20	5	1																			
my.net.5.239	1	1																			
my.net.6.63	11	1																			
my.net.9.31	1	1																			

my.net.14.52	<u>1</u>	1
my.net.24.33	<u>6</u>	1
my.net.24.34	<u>26</u>	6
my.net.24.44	<u>5</u>	1
my.net.24.132	<u>1</u>	1
my.net.24.217	<u>1</u>	1
my.net.25.12	<u>1</u>	1
my.net.25.71	<u>1</u>	1
my.net.25.72	<u>2</u>	2
my.net.27.236	<u>1</u>	1
my.net.29.3	<u>1</u>	1
my.net.29.11	<u>5</u>	1
my.net.41.104	<u>1</u>	1
my.net.41.114	<u>1</u>	1
my.net.42.91	<u>1</u>	1
my.net.53.73	<u>1</u>	1
my.net.60.38	<u>1</u>	1
my.net.60.39	<u>1</u>	1
my.net.65.8	<u>1</u>	1
my.net.68.13	<u>2</u>	1
my.net.69.160	<u>68</u>	1
my.net.69.217	<u>3</u>	1
my.net.70.99	<u>2</u>	1
my.net.71.210	<u>1</u>	1
my.net.72.175	<u>1</u>	1
my.net.74.160	<u>1</u>	1
my.net.74.221	<u>89</u>	1
my.net.76.12	<u>1</u>	1
my.net.82.68	<u>1</u>	1
my.net.84.151	<u>4</u>	2
my.net.87.232	<u>10</u>	3
my.net.94.121	<u>1</u>	1
my.net.97.19	<u>4</u>	1
my.net.97.23	<u>1</u>	1
my.net.97.51	<u>21</u>	1
my.net.97.61	<u>5</u>	1
my.net.97.69	<u>3</u>	1
my.net.97.91	<u>1</u>	1
my.net.97.93	<u>93</u>	1
my.net.97.97	<u>2</u>	1
my.net.97.128	<u>31</u>	1
my.net.97.179	<u>9</u>	1
my.net.98.21	<u>21</u>	1
my.net.98.86	<u>1</u>	1
my.net.101.173	<u>1</u>	1
my.net.107.152	<u>1</u>	1

my.net.110.204	<u>1</u>	1
my.net.111.9	<u>2</u>	1
my.net.111.34	<u>1576</u>	1
my.net.111.78	<u>1</u>	1
my.net.111.197	<u>62</u>	1
my.net.112.141	<u>1</u>	1
my.net.112.195	<u>2</u>	1
my.net.112.196	<u>3</u>	1
my.net.112.226	<u>1</u>	1
my.net.113.4	<u>5</u>	1
my.net.120.220	<u>1</u>	1
my.net.121.37	<u>1</u>	1
my.net.130.16	<u>1</u>	1
my.net.130.143	<u>1</u>	1
my.net.141.21	<u>74</u>	1
my.net.141.176	<u>1</u>	1
my.net.143.66	<u>1</u>	1
my.net.147.150	<u>1</u>	1
my.net.150.83	<u>4</u>	1
my.net.152.173	<u>1</u>	1
my.net.153.187	<u>1</u>	1
my.net.159.101	<u>1</u>	1
my.net.162.53	<u>1</u>	1
my.net.162.191	<u>1</u>	1
my.net.177.204	<u>1</u>	1
my.net.178.69	<u>1</u>	1
my.net.182.104	<u>1</u>	1
my.net.182.158	<u>1</u>	1
my.net.183.50	<u>1</u>	1
my.net.190.99	<u>2</u>	1
my.net.190.201	<u>1</u>	1

Now for the sources that are generating this alert, since we have no content or access to the rule. These could be attempting to propagate the worm or false positives, but we need to check. There is really not enough here to absolutely rule them out as false positives. Possible compromised hosts.

< Src IP address >	< Total # >	< Dest. Addr. >
my.net.97.19	5	1
my.net.97.23	1	1
my.net.97.51	22	1
my.net.97.61	6	1
my.net.97.69	1	1
my.net.97.91	1	1
my.net.97.93	131	1
my.net.97.97	3	1
my.net.97.128	33	1
my.net.97.179	13	1
my.net.98.21	17	1
my.net.98.86	2	1
my.net.100.165	6	1
my.net.100.230	18	4
my.net.110.204	1	1
my.net.111.34	2330	1
my.net.111.197	55	1
my.net.113.4	4	1
my.net.141.21	57	1
my.net.150.83	3	1
my.net.5.20	4	1
my.net.6.15	1	1
my.net.6.55	18	2
my.net.6.63	10	1
my.net.12.4	15	2
my.net.24.20	8	2
my.net.24.33	13	1
my.net.24.34	26	6
my.net.24.44	5	1
my.net.29.3	1	1
my.net.29.11	5	1

my.net.53.73	1	1
my.net.60.17	4	1
my.net.60.38	1	1
my.net.69.160	99	1
my.net.69.217	2	1
my.net.70.99	2	1
my.net.74.221	78	1
my.net.84.151	2	1
my.net.87.232	8	3

Correlations:

Defensive Recommendations: Check hosts that are known to have the vulnerability with the list of destinations and sources. Any matches should be checked for signs of compromise. If compromised rebuild and patch.

[spp_http_decode: CGI Null Byte attack detected](#)⁸

[5170](#)
(2%)

[68](#)

[91](#)

Analysis:

Traffic to hosts that are not running web servers can be ruled out as false positives. A lot of the alerts are also being generated by the dialup traffic going outbound. These are also false positives. After further review it appears that practically all of the alerts are being generated by outbound traffic. They are all false positives and can probably be tuned out.

The only traffic that is inbound:

[67.249.232.199](#) [my.net.24.34](#)

[67.243.133.85](#) [my.net.24.34](#)
[65.214.36.116](#) [my.net.24.34](#)
[68.54.94.58](#) [my.net.24.34](#)

The destination [my.net.24.34](#) is a web server. The content of these packets should be checked. Possible host compromise.

Correlations:

Defensive Recommendations: Rebuild confirmed compromised hosts. Block traffic from above four hosts. To reduce the amount of traffic to look through when analyzing alerts turn off the http_decode preprocessor for the sensors on outbound traffic.

CS WEBSERVER - external ftp traffic	3693 (1%)	148	1
--	------------------------------	---------------------	-------------------

Description: A custom rule to detect port 21 traffic to the CS web server.

Analysis:

This rule appears to be used for logging ftp connections. These alerts are pretty meaningless in the context of intrusion detection without further information as to who is, and, who is not allowed access. Snort is a great packet logger, but the output should probably not be included with the other IDS detects. These are probably all legitimate connections, which should not be part of an Intrusion Detection System.

Correlations:

Defensive Recommendations: Split the web server from the ftp server to be able to segregate the traffic.

my.net.30.3 activity	3584 (1%)	44	1
-----------------------------	------------------------------	--------------------	-------------------

Description: A custom rule that detected traffic to ports 21, 22, 80, 524, 3372, 3389, 3410, 4899, 45000 and 51443.

Analysis:

From the unique TCP destination ports table, we can see that most of the traffic is to ports 80 and 524. 524 is ncp. Probably false positives. The other activity needs more investigation.

≤ Port ≥	≤ Occurrences ≥	≤ Source IP ≥	≤ Dest. IP ≥	≤ First ≥	≤ Last ≥
21 / tcp	1	1	1	2003-07-09 00:57:19	2003-07-09 00:57:19
22 / tcp	2	2	1	2003-07-06 00:59:42	2003-07-06 02:09:29
80 / tcp	82	19	1	2003-07-06 01:15:39	2003-07-09 16:18:14
524 / tcp	3491	17	1	2003-07-05 00:27:43	2003-07-09 22:58:34
3372 / tcp	1	1	1	2003-07-08 21:52:39	2003-07-08 21:52:39
3389 / tcp	3	1	1	2003-07-08 21:55:56	2003-07-08 21:55:57
3410 / tcp	1	1	1	2003-07-09 00:05:46	2003-07-09 00:05:46
4899 / tcp	2	1	1	2003-07-09 16:18:27	2003-07-09 16:18:28
45000 / tcp	1	1	1	2003-07-09 00:48:55	2003-07-09 00:48:55
2003-07-09 00:57:19		212.252.91.20 :54608		my.net.30.3 :21	

2003-07-06 00:59:42	80.55.196.26:58054	my.net.30.3:22
2003-07-06 02:09:29	213.35.200.178:1592	my.net.30.3:22
2003-07-08 21:52:39	195.223.97.170:1496	my.net.30.3:3372
2003-07-08 21:55:56	62.194.170.250:4510	my.net.30.3:3389
2003-07-08 21:55:57	62.194.170.250:4510	my.net.30.3:3389
2003-07-08 21:55:57	62.194.170.250:4510	my.net.30.3:3389
2003-07-09 00:05:46	67.74.154.85:4377	my.net.30.3:3410
2003-07-09 16:18:27	217.120.249.241:1964	my.net.30.3:4899
2003-07-09 16:18:27	217.120.249.241:1965	my.net.30.4:4899
2003-07-09 00:48:55	68.68.249.101:2788	my.net.30.3:45000

All this external activity is very suspicious. Check host for compromise.

Correlations: cc150743-a.assen1.dr.home.nl (217.120.249.241) also appears in alerts my.net.30.4 activity, Notify Brian B. 3.54 tcp , and Notify Brian B. 3.56 tcp.

Defensive Recommendations: Check hosts for compromise. Rebuild confirmed compromised hosts. Turn off unused services. Block external connections. This rule appears to be used for logging connections. Snort is a great packet logger, but the output should probably not be included with the other IDS detects.

EXPLOIT x86 NOOP ⁹	3370 (1%)	42	92
---	------------------------------	--------------------	--------------------

Description: Use above hyperlink. The x86 NOP can frequently be found in day-to-day traffic, particularly when transferring large files.

Analysis:

This alert is known to generate a lot of false positives. Examining the alerts, we find that it is hard to distinguish this from normal web traffic. One of the alerts is to and from news servers, probably a false positive. Not a lot of clues here since we don't have the content. So I used a link graph on the time profile of the alerts. Then a query during the time of highest activity. This search produced 1552 alerts from 1:00 to 12:00 on 07-09-03. A closer look at some of the traffic revealed some odd behaviors that might not be just simple web browsing. A couple of the external hosts (from foreign countries) seemed to stay on a long time (all night) and there source ports and connections appeared different

then most of the other traffic.

07/9/2003 1:00:00 - 1:59:59	32	
07/9/2003 2:00:00 - 2:59:59	191	
07/9/2003 3:00:00 - 3:59:59	1	
07/9/2003 4:00:00 - 4:59:59	181	
07/9/2003 5:00:00 - 5:59:59	1	
07/9/2003 6:00:00 - 6:59:59	325	
07/9/2003 7:00:00 - 7:59:59	101	
07/9/2003 8:00:00 - 8:59:59	248	
07/9/2003 9:00:00 - 9:59:59	0	
07/9/2003 10:00:00 - 10:59:59	444	
07/9/2003 11:00:00 - 11:59:59	28	

Source 217.106.116.202 post.comch.ru

≤ Dest IP address ≥	≤ Total # ≥	≤ Unique Alerts ≥	≤ Src. Addr. ≥
my.net.5.15	8	1	1
my.net.5.44	81	1	1
my.net.5.55	12	1	1
my.net.5.67	116	1	1
my.net.5.92	158	1	1
my.net.5.95	146	1	1

my.net.29.12	155	1	1
my.net.86.19	158	1	1
my.net.163.84	128	1	1
my.net.184.47	127	1	1

Source 213.92.206.11 c206011.adsl.hansenet.de

≤ Dest IP address ≥	≤ Total # ≥	≤ Unique Alerts ≥	≤ Src. Addr. ≥
my.net.5.15	10	1	1
my.net.5.20	17	1	1
my.net.5.44	13	1	1
my.net.5.46	16	1	1
my.net.5.55	8	1	1
my.net.5.67	16	1	1
my.net.5.92	20	1	1
my.net.5.95	25	1	1
my.net.29.8	19	1	1
my.net.29.12	17	1	1
my.net.29.18	15	1	1
my.net.29.19	12	1	1
my.net.86.19	103	1	1
my.net.111.21	19	1	1
my.net.184.25	19	1	1
my.net.184.47	19	1	1

Check the above hosts for signs of compromise.

Correlations:






Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised host. Block external connections.

connect to 515 from inside	3057 (1%)	1	1
-----------------------------------	------------------------------	-------------------	-------------------

Description: A custom rule that detected traffic from a laptop (my.net.162.42) on port 721 to a (tektronics printer?) (tek924.gsfc.nasa.gov) on port 515.

Analysis:

There are a lot of connections here. It could be printing, all of the activity occurs during the day or early evening. For example, the following link graph shows a timeline on 07/07/03.

07/7/2003 9:00:00 - 9:59:59	2	
07/7/2003 10:00:00 - 10:59:59	54	
07/7/2003 11:00:00 - 11:59:59	81	
07/7/2003 12:00:00 - 12:59:59	47	
07/7/2003 13:00:00 - 13:59:59	21	

07/7/2003 14:00:00 - 14:59:59	0	
07/7/2003 15:00:00 - 15:59:59	92	
07/7/2003 16:00:00 - 16:59:59	136	
07/7/2003 17:00:00 - 17:59:59	91	
07/7/2003 18:00:00 - 18:59:59	88	
07/7/2003 19:00:00 - 19:59:59	145	

It is a possible false positive. Need more information. Is this host running Linux? If so, check host for signs of compromise.

Correlations:

Defensive Recommendations: Check for signs of compromise. Rebuild confirmed promised host. Block external connections.

External RPC call ¹⁰	2539 (1%)	4	1138
---	------------------------------	-------------------	----------------------

Description: Use above hyperlink.

Analysis:

66.198.148.9 is scanning the my.net.133.0/24 134, 135, 137, 190 subnets with source and destination port of 110. Probably using a script to automate it. Information gained from these scans could be used for more active targeting later.

	Total	Dest. Addr.
66.198.148.9	863	849
195.13.253.73	193	188
211.114.9.211	1096	1044
211.168.183.66	387	376

2003-07-06 22:45:39	211.114.9.211 :4354	my.net.135.44 :111
2003-07-06 22:45:39	211.114.9.211 :4355	my.net.135.45 :111
2003-07-06 22:45:39	211.114.9.211 :4356	my.net.135.46 :111
2003-07-06 22:45:39	211.114.9.211 :4358	my.net.135.48 :111
2003-07-06 22:45:39	211.114.9.211 :4357	my.net.135.47 :111
2003-07-06 22:45:39	211.114.9.211 :4359	my.net.135.49 :111

In this part of the scan you can see the incrementing source ports. 211.114.9.211 is scanning, the 132, 133, 134, 135 subnets. I don't believe this is worm activity, but you never know. It might be prudent to check some of the vulnerable hosts for signs of compromise.

A self-propagating Linux worm called Ramen has been reported. ¹¹ This worm is known to infect Red Hat 6.2 and 7.0 machines. It infects the machines with vulnerabilities in wu-ftp, rpc.statd, and LPRng services. This worm has the ability to infect other Linux and

Unix machines via a vulnerable wu-ftp version, rpc.statd and LPRng. This worm can also be easily modified since it leaves the source code on the machine. GIAC has received several reports of this worm infecting machines, and the network traffic that it creates.

The worm uses a tool called synscan which has been modified to fit its needs. Using this tool, the worm contacts a randomly generated IP address and checks the FTP banner to determine if the machine is running Red Hat Linux 6.2 or Red Hat Linux 7.0. For machines running Red Hat 6.2, the worm will attempt to exploit a vulnerable rpc.statd or wuftpd service. For Red Hat 7.0, the worm tries to exploit an LPRng bug to gain access to the system.

Correlations:

Defensive Recommendations: Limit remote access to RPC services. Filter RPC ports at the firewall to ensure access is denied to RPC-enabled machines. Disable unneeded RPC services.

High port 65535 udp - possible Red Worm – traffic ¹²	1527 (1%)	75	73
--	------------------------------	--------------------	--------------------

Description:

Code Red supposedly opens a backdoor on this port. This is accomplished by copying the standard Windows NT/2000 command interpreter "cmd.exe" into web server's "scripts" directory. The worm drops a trojan program to 'explorer.exe' that modifies different some IIS settings to allow a remote attack of the infected host. The standard command interpreter 'cmd.exe' is copied to 'inetpub\scripts\root.exe' and to 'progra~1\common~1\system\MSADC\root.exe'. The worm creates these files to both 'C:' and 'D:' drives if they exist. These copies of the 'cmd.exe' will allow any attacker to execute commands on the remote system really easily.

Analysis: Appears to be a custom rule that alerts on UDP traffic with a source or destination port of 65535. It is not known if this rule is just a protocol rule, or whether it involves content also. Once again there is not enough information to weed out the false positives. So we need to check all the source and destination hosts that set this rule off. We need to check each source or destination against the list of known hosts with the vulnerability. The following are possible compromised hosts.

Sources

≤ Src IP address ≥	≤ Total# ≥	≤ Dest.Addr. ≥
my.net.70.164	2 2	
my.net.72.249	12 1	
my.net.84.151	2 1	
my.net.84.178	28 2	
my.net.86.110	263 12	
my.net.97.23	1 1	
my.net.97.69	1 1	
my.net.97.102	5 1	
my.net.97.134	1 1	
my.net.111.34	12 1	

my.net.150.242	<u>56</u>	5
my.net.153.113	<u>1</u>	1
my.net.153.223	<u>327</u>	13

Destinations

\leq Dest IP address \geq	\leq Total# \geq	\leq Src.Addr. \geq
my.net.53.35	<u>1</u>	1
my.net.69.136	<u>2</u>	1
my.net.69.217	<u>89</u>	2
my.net.69.249	<u>7</u>	1
my.net.70.164	<u>2</u>	2
my.net.72.249	<u>4</u>	1
my.net.74.247	<u>1</u>	1
my.net.80.15	<u>2</u>	1
my.net.81.112	<u>2</u>	1
my.net.84.150	<u>1</u>	1
my.net.84.178	<u>52</u>	10
my.net.84.198	<u>1</u>	1
my.net.86.110	<u>260</u>	12
my.net.97.23	<u>1</u>	1
my.net.97.69	<u>1</u>	1
my.net.97.102	<u>5</u>	1
my.net.97.110	<u>1</u>	1
my.net.97.134	<u>1</u>	1
my.net.98.103	<u>1</u>	1
my.net.108.34	<u>1</u>	1
my.net.111.34	<u>8</u>	1
my.net.111.139	<u>2</u>	1
my.net.114.110	<u>3</u>	3
my.net.117.10	<u>1</u>	1
my.net.150.121	<u>18</u>	1
my.net.150.203	<u>5</u>	1
my.net.150.242	<u>24</u>	11
my.net.151.115	<u>4</u>	1
my.net.152.14	<u>1</u>	1
my.net.152.162	<u>3</u>	1
my.net.152.163	<u>11</u>	1
my.net.153.105	<u>1</u>	1
my.net.153.113	<u>4</u>	2
my.net.153.185	<u>1</u>	1
my.net.153.223	<u>283</u>	11
my.net.198.244	<u>2</u>	1

Correlations:

Defensive ecommendations: Check hosts that are known to have the vulnerability with the list of sources and destinations. Any matches should be checked for signs of compromise. If compromised rebuild and patch.

TCP SRC and DST outside network	<u>1192</u> (0%)	<u>90</u>	<u>326</u>
--	-------------------------------------	---------------------------	----------------------------

Description: A custom rule that detected traffic that is not bound for or did not originated on this network, 90 source addresses with 326 destination addresses.

Analysis:

The sources are mostly from aol.com. What is this doing here? If these addresses are not spoofed, there is a major problem in that external traffic is traversing the internal LANs, needs to be investigated. Could indicate spoofing on the internal network or poor network design.

Correlations:

Defensive Recommendations: Implement egress filtering, if traffic that is leaving the LAN does not have a source address of that LAN it is dropped. If external traffic is allowed to cross the internal LAN, the source should be found and eliminated.

<u>IDS552/web-iis IIS ISAPI Overflow ida nosize</u>	<u>890</u> (0%)	<u>619</u>	<u>374</u>
---	---------------------------------	----------------------------	----------------------------

Description: Use above hyperlink.

Analysis:

Supposedly there are no known false positives with this alert. Given that assumption we will assume all the alerts are genuine and this is the worm attempting to replicate. Given the number of alerts and the fact that there is no active targeting go on, we won't attempt to analyze each but instead, we should compile a list of internal hosts with the vulnerability. Once we have the list then we can see if an alert was generated for it and proceed from there. Given that, it looks like we have possible compromised hosts.

Correlations:

Defensive Recommendations: If a vulnerable host has a detect generated, check for signs of compromise. Rebuild compromised hosts. Patch all vulnerable hosts. Block external connections.

Null scan! ¹³	<u>775</u> (0%)	<u>32</u>	<u>38</u>
---------------------------------	------------------------------------	---------------------------	---------------------------

Description: This is a TCP probe with no flags set. If sent to an open port a RST/ACK is received, for a closed port nothing is received. It can also be used for OS fingerprinting. If a NULL scan shows all ports closed while a SYN scan shows open ports, the host is one of the following: Windows, CISO, BSDI, HP/UX, MVS, IRIX. As part of information gathering leading up to another (more directed) attack, an attacker may attempt to figure out what ports are open/closed on a remote machine.

Analysis:

Top four scanners

	Total	Dest Addr
<u>213.176.8.2</u>	<u>364</u>	4
		<u>my.net.25.69</u>

		my.net.25.71 my.net.25.72 my.net.25.73		
67.119.233.217	268	5		
		my.net.12.4:110 my.net.25.21:110 my.net.25.22:110 my.net.25.23:110 my.net.25.24:110		
These are mail servers looking to see if pop-3 is running				
141.156.139.19	54	3		
		my.net.12.2 my.net.25.10 my.net.25.12		
63.251.52.75	36	4		
		my.net.153.114:0 my.net.150.68:0 my.net.178.66:0 my.net.114.115:0		
<p>Supposedly there are no known false positives for this signature, so these hosts are the victims of network reconnaissance, probably will be actively targeted in the future.</p> <p>Correlations: 63.251.52.75 Probable NMAP fingerprint attempt.</p> <p>Defensive Recommendations: Determine if this particular port would have responded as being open or closed. If open, watch for more attacks on this particular service or from the remote machine that sent the packet. If closed, simply watch for more traffic from this host. ¹⁴ Block external connections.</p>				
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize		685 (0%)	2	473
Description: See above.				
Analysis:				
		Total	Dest Addr.	
my.net.69.145	539		378	
Time			#Alerts	
07/5/2003 0:00:00 - 0:59:59			380	
07/5/2003 1:00:00 - 1:59:59			159	
my.net.97.61	146		95	
Time			#Alerts	
07/5/2003 1:00:00 - 1:59:59			146	
This looks like the activity of a host infected by a worm.				
Correlations: my.net.97.61 513 occurrences as src 9 as dest my.net.69.145 960 occurrences as src 41 as dest				
Defensive Recommendations: Apply the patch. Block access from external connections. Check for signs of compromise. Rebuild confirmed compromised hosts.				
NMAP TCP ping! ¹⁵		595	155	61

		(0%)		
Description: Sometimes you may merely want to check the availability of a system without sending ICMP echo requests, which may be blocked by some sites. In this case, a TCP "ping" sweep can be used to scan a target's network.				
Analysis:				
155 sources to 61 destinations. The top three destinations (name servers) are shown below				
	Total	Src Addr.		
my.net.1.3	171	67		
my.net.1.4	50	26		
my.net.1.5	16	14		
Top sources	Total	Dest. Addr.		
63.211.17.228	77	17		
64.152.70.68	67	14		
193.41.181.254	76	1		
This is most likely network reconnaissance in preparation for future attacks.				
Correlations:				
Defensive Recommendations: Block external connections.				
connect to 515 from outside		518 (0%)	1	1
Description: The alerts are triggering on a source port of 721 and a destination port of 515. This is the UNIX printer daemon. A known vulnerability exists: CVE-2000-0917 ¹⁶				
Format string vulnerability in use_syslog() function in LPRng 3.6.24 allows remote attackers to execute arbitrary commands.				
tcp any 515				
Redhat/Mandrake uses LPRng. The Code Red worm scans for this as well as the Ramen worm.				
Analysis:				
2003-07-08 09:05:50 131.118.229.7 :721 my.net.24.15 :515				
518 alerts to newprinter, false positive.				
Correlations:				
Defensive Recommendations: Tune IDS. Verify host is a printer. Block external connections.				
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan. ¹⁷		464 (0%)	72	62
Description: Custom rule that detects the kill command. The kill command causes the specified client-server connection to be closed by the server that has the actual connection. The comment parameter reflects the reason the connection was killed. This command can only be executed by an IRC operator. The Console window displays the results of this command. The syntax for the KILL command is as follows:				
/KILL <nickname><comment>				
Analysis:				
Unique TCP Source ports				

<u>≤ Port ≥</u>	<u>≤ Occurrences ≥</u>	<u>≤ Source IP ≥</u>	<u>≤ Destination IP ≥</u>	<u>≤ First ≥</u>	<u>≤ Last ≥</u>
6660 / tcp	16	1	1	2003-07-09 11:02:23	2003-07-09 23:28:09
6661 / tcp	28	6	1	2003-07-09 10:31:45	2003-07-09 23:11:50
6662 / tcp	1	1	1	2003-07-09 01:04:04	2003-07-09 01:04:04
6663 / tcp	18	5	1	2003-07-09 10:29:12	2003-07-09 23:46:28
6665 / tcp	32	8	3	2003-07-07 08:00:57	2003-07-09 23:35:23
6666 / tcp	17	7	4	2003-07-07 07:59:51	2003-07-09 16:50:48
6667 / tcp	291	60	50	2003-07-05 00:26:13	2003-07-09 23:24:18
6668 / tcp	13	2	2	2003-07-09 09:58:55	2003-07-09 23:37:29
6669 / tcp	36	4	2	2003-07-08 16:27:37	2003-07-09 23:36:26
6670 / tcp	1	1	1	2003-07-08 20:10:14	2003-07-08 20:10:14
7000 / tcp	11	6	11	2003-07-06 00:26:16	2003-07-09 00:59:23

From the query we can see that all of the alerts are using well-known Trojan ports. There are 72 source hosts and 62 destination hosts. There does not appear to be any traffic to well-known ports, which could be false positives. We don't have the rule so we can't evaluate the possibilities of false positives. However a rule that detects this traffic if it involves content should be fairly easy to write, so that the incidence of false positives would be minimum. Probably no false positives. Possible Trojans controlled from IRC. All internal hosts should be checked for compromise.

Correlations:

Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised hosts. Block access to external connections.

Possible trojan server activity ¹⁸	445 (0%)	44	55
--	-----------------------------	--------------------	--------------------

Description: All detects are triggering on port 27374. Detects to recognizable ports may be false positives. The following Trojans use this port: Sub Seven 2.1 (UDP), Bad Blood, Ramen, Seeker, Sub Seven 2.14, Sub Seven, Muie, DefCon 8, Ttfloader. Port 1214 Kaaza

Analysis:

Traffic on ports 22,25,80,110,143,443 is most likely false positives.

2003-07-08 18:51:07	my.net.113.4:1214	62.202.2.3:27374
2003-07-08 18:51:08	my.net.113.4:1214	62.202.2.3:27374
2003-07-06 11:04:28	my.net.113.4:1214	208.180.101.103:27374

Using Kaaza ports needs further investigation.

2003-07-08 04:46:48	my.net.84.235 :27374	80.218.101.63 :4662
2003-07-08 04:46:54	my.net.84.235 :27374	80.218.101.63 :4662
2003-07-09 00:53:05	my.net.84.235 :27374	213.93.208.223 :4662
2003-07-09 00:53:08	my.net.84.235 :27374	213.93.208.223 :4662
2003-07-09 00:53:14	my.net.84.235 :27374	213.93.208.223 :4662

Using well known Trojan ports. Possible compromise.

Correlations: Host my.net.84.235 also appears in the following alerts: spp_http_decode: IIS Unicode attack detected, spp_http_decode: CGI Null Byte attack detected, DDOS shaft client to handler. Host my.net.113.4 also appears in the following alerts: High port 65535 tcp - possible Red Worm – traffic, Queso fingerprint , Incomplete Packet Fragments Discarded and SMB Name Wildcard.

Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised hosts. Block access to external connections.

NIMDA - Attempt to execute cmd from campus host¹⁹	409 (0%)	8	397
---	-----------------------------	-------------------	---------------------

Description: Unlike the Code Red II Worm, the Nimbda worm requests for a "cmd.exe" file in various subdirectories of your server, and affects more than just Microsoft's IIS web servers: it also exploits holes in Microsoft Outlook, Outlook Express and Internet Explorer. Sends itself by email. Searches for open network shares. Attempts to copy itself to unpatched or already vulnerable Microsoft IIS web servers. Is a virus infecting both local files and files on remote network shares. The worm uses the Unicode Web Traversal exploit.

Analysis:

	Total	Dest Addr
my.net.10.177	1	1
my.net.30.86	1	1
my.net.69.145	318	316
my.net.97.40	1	1
my.net.97.61	85	79
my.net.97.176	1	1
my.net.132.45	1	1
my.net.191.67	1	1

Two of the external hosts are exhibiting worm like behavior. Need to check these 8 hosts to determine if they have the vulnerability. Possible compromised hosts.

Correlations:

Defensive Recommendations: Clean or rebuild confirmed compromised hosts. Apply the patch. Block access from external connections.

SUNRPC highport access!	343 (0%)	15	15
--------------------------------	-----------------------------	--------------------	--------------------

Description: See above External RPC call or below Attempted Sun RPC high port access. Custom rule that alerted on internal destinations using port 32771.

Analysis: The traffic between the mail servers, the web traffic to google.com, cnn.com, real.com, kernel.org, terraplex.com, and naver.com the ssl traffic to redhat network are

most likely all probably a false positives. The traffic to xserve.blender.org is probably a false positive also.

2003-07-06 12:11:46 [64.12.161.153](#):5190 [my.net.97.89](#):32771

2003-07-06 12:11:47 [64.12.161.153](#):5190 [my.net.97.89](#):32771

and with 106 alerts (only two shown)

2003-07-08 12:15:13 [205.188.7.200](#):5190 [my.net.69.254](#):32771

2003-07-08 12:15:39 [205.188.7.200](#):5190 [my.net.69.254](#):32771

These two hosts are probably using AOL instant messenger, false postive.

2003-07-09 17:40:41 [205.188.234.3](#):8030 [my.net.70.234](#):32771

2003-07-09 17:40:41 [205.188.234.3](#):8030 [my.net.70.234](#):32771

The last host my.net.70.234 deserves further attention.

Correlations:

Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised hosts. Block access to external connections.

SMB C access ²⁰	310 (0%)	65	104
--	-----------------------------	--------------------	---------------------

Description: A remote user has attempted to access the C\$ default administrative share of a Windows host. **Analysis:**

A very large number of detects for this signature. The number of source and destination addresses is very disturbing. Since this attack can lead to administrator access on the local host. On second look maybe not so bad. We have a scan from [67.35.116.205](#):21103 [my.net.152.183](#):139 this accounts for 179 of the alerts. So this source could be doing reconnaissance. Targeting the my.net.152.0/24 and my.net.153.0/24 subnet hosts for future attacks.

Unique destination addresses for this scan

	Total	Src Addr
my.net.132.45	35	14
my.net.137.34	23	14
my.net.137.36	9	9
my.net.137.37	7	7
my.net.137.46	11	11
my.net.190.19	4	4
my.net.190.100	20	20
my.net.190.102	22	22

The above eight hosts should be checked for possible compromise

Correlations: 67.35.116.205 has 607 occurrences of which 428 are SMB Name Wildcard.

Defensive Recommendations: Check for signs of compromise. Rebuild confirmed compromised hosts. Disallow Netbios access from external networks (tcp port 139).

SNMP public access ²¹	119 (0%)	1	1
---	-----------------------------	-------------------	-------------------

Description: This means that snort saw a SNMP packet (used to administer devices

such as switches) accessing (or attempting to access) the "public" SNMP community, a common default setting in devices. (Actually, it triggers on any snmp packet containing the word "public".)

Based on what you show of IP addresses, this is traffic within your network, and is probably normal. I'd be concerned if either machine wasn't one I controlled. I'd also consider changing my SNMP community strings to something other than public.

SNMP (Simple Network Management Protocol) v1 uses communities and IP addresses to authenticate communication between the SNMP client and SNMP daemon. Many SNMP implementations come pre-configured with 'public' and 'private' communities. If these are not disabled, the attacker can gather a great deal of information about the device running the SNMP daemon. An attacker scans a range of IPs for SNMP servers having the 'public' community set and gathers information about the hosts.

Analysis:

2003-07-08 07:51:59 [134.192.86.65](#):1047 [my.net.190.13](#):161

It looks like the host 134.192.86.65 has walked the snmp tree for my.net.190.13. This is an information leak.

Correlations:

Defensive Recommendations: Disable the 'public' and 'private' communities before connecting the device with SNMP on the Internet or block access to SNMP ports using a packet filtering firewall for unauthorized addresses.

NIMDA - Attempt to execute root from campus host

[117](#)
(0%)

[2](#)

[116](#)

Description: See above.

Analysis:

The activity proceeds as follows

My.net.69.145 start 2003-07-05 01:00:46 stop 2003-07-05 01:12:05

103 destination addresses are almost exclusively from the 130.223.0.0/16 subnets.

My.net.97.61 start 2003-07-05 01:15:10 stop 2003-07-05 01:20:29

13 destination addresses mostly to 130.223.0.0/16 subnets. Looks like worm activity.

Check these two hosts for compromise.

Correlations:

Defensive Recommendations: Check for signs of compromise. Rebuild compromised hosts. Apply the patch. Block access from external connections.

Incomplete Packet Fragments Discarded²³

[114](#)
(0%)

[41](#)

[29](#)

Description: This message is given by the defragmentation preprocessor when packets bigger than 8k that are more than half empty when the last fragment is received are discarded.

This can be caused by:

- transmission errors

- broken stacks
- and fragmentation attacks

Analysis:

Probably not fragmentation attacks. Most of the source and destination addresses are dialup and DSL. Transmission errors, seems a good bet, since these types of connections tend to be noisier and more error prone. Dip.t-dialin.net in particular has the majority of alerts. Most are probably false positives.

Some other interesting traffic is:

61-25-140-151.home.ne.jp [61.25.140.151](#) --> [my.net.112.196](#)
 xdsl-195-14-219-161.netcologne.de [195.14.219.161](#) --> [my.net.97.114](#)
 p50854a7f.dip0.t-ipconnect.de [80.133.74.127](#) --> [my.net.100.165](#)
 afontenayssb-111-1-1-247.w80-13.abo.wanadoo.fr [80.13.0.247](#) -->
[my.net.112.196](#)
 afontenayssb-110-1-4-88.w81-48.abo.wanadoo.fr [81.48.183.88](#) -->
[my.net.111.34](#)
 r200-40-184-73.adsl.anteldata.net.uy [200.40.184.73](#) --> [my.net.150.220](#)

Correlations:

Defensive Recommendations: Block external connections.

TFTP - Internal TCP connection to external tftp server	114 (0%)	7	65
---	-----------------------------	-------------------	--------------------

Description: Custom rule that alerts on TCP traffic to a source or destination port of 69 and an external address.

Analysis:

Unique TCP Source Addresses

< Src IP address >	≤ Total # ≥	≤ Unique Alerts ≥	≤ Dest. Addr. ≥
my.net.18.47	2	1	2
my.net.97.181	12	1	1
my.net.162.67	1	1	1
140.239.42.26	1	1	1
198.64.149.228	60	1	51
198.173.255.237	34	1	30
216.17.103.14	4	1	1

This traffic needs to be looked at closely. It could be compromised hosts attempting to download Trojans, rootkits and other tools. supershe.tempdomainname.com 198.173.255.237 has 8 other occurrences as a destination (TCP SRC and DST outside network). They are telnets and ssh from [172.137.152.210](#). This is very suspicious. Can't really rule out any of the internal hosts as not possible sources of unauthorized activity. Check all internal hosts involved in this rule for signs of possible compromise.

Correlations: my.net.97.81 also appears in spp_http_decode: IIS Unicode attack detected. my.net.162.67 also appears as a destination in [IDS552](#)/web-iis_IIS ISAPI Overflow ida nosize. my.net.18.47 also appears in spp_http_decode: IIS Unicode attack detected.

Defensive Recommendations: Rebuild confirmed compromised hosts. Turn off and

remove unneeded services. Block external connections.

IRC evil - running XDCC

79
(0%)

3

3

Description: Custom rule to detect XDCC bots. All alerts have a destination port of 6667 or 6669, the source ports are 1026,1036, and 1986. See description below (User joining XDCC channel detected. Possible XDCC bot²⁴ and Possible Incoming XDCC Send Request Detected).

Analysis:

[my.net.80.209](#) --> [66.207.164.23](#)
[my.net.74.216](#) --> [212.161.35.251](#)
[my.net.198.221](#) --> [205.188.149.12](#)

Possible compromised hosts running XDCC. Possible Trojans. Known Trojans ports.

Correlations: 212.161.35.251 has 10 occurrences as source and 47 occurrences as a destination). It appears in the alerts below, [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected and [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC.

Defensive Recommendations: Rebuild confirmed compromised hosts. Turn off and remove unneeded services. Enforce a password policy. Block external connections.

EXPLOIT x86 setuid 0²⁵

58
(0%)

40

26

Description: Use above hyperlink.

Analysis:

2003-07-08 00:18:47	144.118.198.20:1901	my.net.111.51:3297
2003-07-08 00:31:10	144.118.198.20:1915	my.net.111.51:3299
2003-07-08 00:48:50	144.118.198.20:1936	my.net.111.51:3303
2003-07-08 00:48:58	172.153.29.188:6699	my.net.97.56:2253
2003-07-08 01:04:50	209.214.160.100:6699	my.net.97.56:2144
2003-07-08 01:09:36	144.118.198.20:1950	my.net.111.51:3304
2003-07-08 01:53:58	63.240.202.73:4000	my.net.97.64:1303
2003-07-08 02:10:34	66.229.65.137:6699	my.net.97.56:2474
2003-07-08 02:11:50	63.240.202.38:4000	my.net.97.64:1305
2003-07-08 02:18:09	63.240.202.59:4000	my.net.97.64:1307
2003-07-08 02:19:33	63.240.202.59:4000	my.net.97.64:1307
2003-07-08 02:20:29	63.240.202.59:4000	my.net.97.64:1307
2003-07-08 02:29:40	63.240.202.161:4000	my.net.97.64:1309
2003-07-08 03:04:09	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:04:18	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:04:26	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:07:04	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:07:07	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:07:42	63.240.202.92:4000	my.net.97.64:1336
2003-07-08 03:53:48	63.240.202.37:4000	my.net.97.64:1352
2003-07-08 03:53:52	63.240.202.37:4000	my.net.97.64:1352
2003-07-08 03:54:00	63.240.202.37:4000	my.net.97.64:1352
2003-07-08 04:00:11	63.240.202.174:4000	my.net.97.64:1354
2003-07-08 04:09:21	63.240.202.168:4000	my.net.97.64:1360
2003-07-08 05:46:11	198.118.229.166:58577	my.net.163.97:22
2003-07-08 07:08:42	209.217.135.199:80	my.net.17.48:1920
2003-07-08 08:22:05	64.4.25.188:80	my.net.97.101:1107

2003-07-08 08:27:59	64.4.25.188:80	my.net.97.101:1189
2003-07-08 09:32:42	63.240.202.70:4000	my.net.97.170:1402
2003-07-08 16:08:00	66.230.135.50:80	my.net.190.101:1702
2003-07-08 20:07:01	64.250.215.245:4388	my.net.69.160:3108
2003-07-09 00:09:45	131.118.254.130:1547	my.net.24.8:119
2003-07-09 03:14:54	131.118.254.130:1591	my.net.24.8:119
2003-07-09 07:57:34	64.246.37.28:80	my.net.83.88:1349
2003-07-09 12:47:58	219.133.7.40:60121	my.net.69.148:6882
2003-07-09 13:02:32	216.169.198.9:15372	my.net.150.133:6881
2003-07-09 13:08:03	131.118.254.130:1749	my.net.24.8:119
2003-07-09 15:10:15	130.94.25.53:49791	my.net.82.101:1611
2003-07-09 15:14:32	64.236.34.141:80	my.net.115.145:50645
2003-07-09 19:29:54	63.250.195.10:0	my.net.153.113:0
2003-07-07 09:59:27	66.118.165.60:80	my.net.153.146:2651
2003-07-07 13:07:10	12.238.126.94:2418	my.net.97.35:1745
2003-07-07 13:07:24	12.238.126.94:2418	my.net.97.35:1745
2003-07-07 13:07:37	12.221.190.60:2308	my.net.97.35:1740
2003-07-07 15:30:50	66.250.223.36:80	my.net.162.188:4200
2003-07-07 15:48:27	141.150.121.159:21096	my.net.153.152:1914
2003-07-07 16:02:59	61.133.84.149:3317	my.net.112.195:3682
2003-07-07 16:10:00	12.239.251.46:1271	my.net.153.152:3082
2003-07-07 17:10:30	64.132.47.105:11158	my.net.152.19:6970
2003-07-06 02:28:10	131.118.254.130:2795	my.net.24.8:119
2003-07-06 10:55:30	140.254.73.38:2843	my.net.84.22:1804
2003-07-06 11:04:44	24.126.115.181:3303	my.net.97.164:2927
2003-07-06 13:42:24	63.240.202.73:4000	my.net.97.187:1407
2003-07-06 14:45:09	63.240.202.77:4000	my.net.97.27:1482
2003-07-06 16:00:05	63.240.202.164:4000	my.net.97.27:1535
2003-07-06 16:39:45	63.240.202.65:4000	my.net.97.27:1549
2003-07-06 17:04:43	63.240.202.166:4000	my.net.97.27:1559
2003-07-06 17:15:59	63.240.202.43:4000	my.net.97.27:1561

Since we don't have the content of the packets we will have to use other clues to determine which detects are false positives and which might be actual attacks. All 58 of the alerts above are shown with 48 sources and 256 destinations. A majority of the destinations are dialup. The port 80 and 119 traffic are most likely false positives. The 24 alerts with port 4000 all come from the same subnet. Port 4000 is a known Trojan port so this traffic is suspect. The port 0 traffic is anomalous and comes from a source we have seen in many other alerts. The rest of the traffic uses odd combinations of source/destination ports and is anomalous. These hosts also should be checked for compromise.

Correlations:

l8.cache.vip.dal.yahoo.com [63.250.195.10](#) it has 80 other occurrences as source.

Defensive Recommendations: Possible compromised hosts. Rebuild confirmed compromised hosts. Block external connections.

[EXPLOIT x86 stealth noop](#)²⁶

[52](#)
(0%)

[4](#)

[4](#)

Description: Use above hyperlink.

Analysis:

news.ums.edu [131.118.254.130](#) --> [my.net.24.8](#)

haven.net.umd.edu [128.8.5.30](#) --> [my.net.24.8](#)
 This is most likely normal news traffic that is triggering the pattern, false positive.

n218103242103.netvigator.com [218.103.242.103](#) --> [my.net.69.148](#)

2003-07-08 12:57:53	218.103.242.103 :1569	my.net.69.148 :3434
2003-07-08 13:04:37	218.103.242.103 :1615	my.net.69.148 :3872
2003-07-08 13:06:41	218.103.242.103 :1625	my.net.69.148 :3969
2003-07-08 13:11:15	218.103.242.103 :1651	my.net.69.148 :4177
2003-07-07 19:02:48	218.103.242.103 :2304	my.net.69.148 :2031
2003-07-07 19:31:11	218.103.242.103 :2461	my.net.69.148 :2627
2003-07-07 19:57:11	218.103.242.103 :2600	my.net.69.148 :4055
2003-07-07 20:39:54	218.103.242.103 :2790	my.net.69.148 :2494
2003-07-07 20:39:56	218.103.242.103 :2790	my.net.69.148 :2494
2003-07-07 20:41:43	218.103.242.103 :2796	my.net.69.148 :2516
2003-07-07 21:24:07	218.103.242.103 :2995	my.net.69.148 :3558

This traffic appears to be a genuine attack the source and destination ports are odd. The timing appears right to. The attacker tries an exploit, waits for results, tunes and tries again. There is even a 6-hour break taken. My.net.69.148 is possibly a compromised host.

2003-07-09 10:58:04	199.89.199.30 :80	my.net.53.93 :2391
2003-07-09 20:43:23	199.89.199.30 :80	my.net.152.159 :2707

Most likely normal web traffic, false positive.

Correlations:

Defensive Recommendations: Possible compromised host Rebuild confirmed compromised host. Block external connections.

FTP passwd attempt ²⁷	52 (0%)	28	2
--	----------------------------	--------------------	-------------------

Description: Use above hyperlink.

Analysis:

All alerts are to one host my.net.24.47 with some external sources setting of multiple detects. Probably a lot of false positives, with a few actual attempts. Possible compromised host.

Correlations:

Defensive Recommendations: Identify, the downloaded file and confirm that it indeed is a valid system password file. Change the user passwords on the system and notify the users. Check for other activity indicative of system compromise. Rebuild if compromised. Ensure that FTP access to sensitive system files is not allowed.

Tiny Fragments - Possible Hostile Activity ²⁸	44 (0%)	7	8
--	----------------------------	-------------------	-------------------

Description: Use above hyperlink

Analysis:

Unable to resolve address [211.196.113.222](#) --> [my.net.97.154](#)

Unable to resolve address [24.197.158.136](#) --> [my.net.84.178](#)
 adsl-141-156-52-91.ba-dsg.net [141.156.52.91](#) --> [my.net.24.58](#)
 adsl-141-156-52-91.ba-dsg.net [141.156.52.91](#) --> [my.net.29.11](#)
 YahooBB219173044051.bbtec.net [219.173.44.51](#) -->
 router2.fdi1-unet.ocn.ne.jp [219.166.32.226](#) --> [my.net.25.73](#)
[61.171.249.46](#) --> [my.net.25.70](#)
 ucommons-114-120.pooled.umbc.edu [my.net.114.120](#) --> [217.224.247.198](#)

Traffic to my.net.25.70 (the mail server) has the most fragments with a count of 29. One of the internal hosts my.net.114.120 is also doing this. These hosts all need to be checked since this type of fragmentation is not typically seen in the wild. The dialups and DSL connection might experience some fragmentation but not packets that are less than 25 bytes. And these are not the last fragments, either. It is most likely being used to evade an, IDS or firewall.

Correlations:

Defensive Recommendations: Possible compromised hosts. Rebuild confirmed compromised hosts.

TFTP - Internal UDP connection to external tftp server	44 (0%)	4	9
---	----------------------------	-------------------	-------------------

Description: Custom rule that alerts on UDP traffic to a source or destination port of 69 and an external address.

Analysis:

Unable to resolve address [208.153.50.192](#) --> [my.net.151.115](#)
 corp-gw.viva.yellowbrix.net [64.125.197.7](#) --> [my.net.1.3](#)
 l8.cache.vip.dal.yahoo.com [63.250.195.10](#) --> [my.net.150.121](#)
[my.net.5.92](#) --> [12.211.236.2](#)
 l8.cache.vip.dal.yahoo.com [63.250.195.10](#) --> [my.net.153.113](#)
 l8.cache.vip.dal.yahoo.com [63.250.195.10](#) --> [my.net.152.250](#)
 l8.cache.vip.dal.yahoo.com [63.250.195.10](#) --> [my.net.153.105](#)
[my.net.5.92](#) --> [81.171.2.192](#)
 l8.cache.vip.dal.yahoo.com [63.250.195.10](#) --> [my.net.150.203](#)

Transferring files is very suspicious activity. Possible compromised hosts.

Correlations: 63.250.195.10 appears in many alerts.

Defensive Recommendations: Check hosts for compromise. Rebuild confirmed compromised hosts. Block external connections.

EXPLOIT x86 setgid 0 ²⁹	40 (0%)	28	33
--	----------------------------	--------------------	--------------------

Description: Use above hyperlink.

Analysis:

2003-07-08 00:44:36 [69.27.66.247](#):19328 [my.net.70.207](#):12203
 2003-07-09 07:42:39 [202.177.192.68](#):14676 [my.net.75.108](#):6970
 2003-07-09 10:02:27 [140.254.73.38](#):4832 [my.net.84.22](#):1804
 2003-07-09 13:03:20 [216.169.198.9](#):15372 [my.net.150.133](#):6881
 2003-07-09 14:35:53 [130.94.25.53](#):49649 [my.net.82.101](#):1294

2003-07-09 16:17:21	24.65.26.168 :60140	my.net.69.148 :1139
2003-07-07 18:39:05	66.222.141.62 :12203	my.net.70.207 :12203
2003-07-07 20:54:38	202.102.188.45 :2993	my.net.163.78 :1837
2003-07-07 21:36:19	131.118.254.130 :1084	my.net.24.8 :119
2003-07-07 23:36:38	24.208.236.11 :64946	my.net.69.160 :3108
2003-07-06 09:41:11	131.118.254.130 :2929	my.net.24.8 :119
2003-07-06 18:25:14	194.47.165.108 :50704	my.net.17.70 :5059

Traffic to web servers is probably legitimate and is probably a false positive. The traffic on the high-ports is suspicious and needs further investigation. Possible compromised hosts.

Correlations:

Defensive Recommendations: Check if hosts have the vulnerability, confirm compromise, and rebuild as necessary. Block external connections. Funnel outbound web traffic through an application-level proxy firewall and perform network address translation on it.

ICMP SRC and DST outside network	35 (0%)	9	21
---	----------------------------	-------------------	--------------------

Description: A custom rule that detects ICMP not bound for or originating from this network.

Analysis:

20 of the alerts come from one source ([68.48.32.16](#)), bound for dsl on aol.com, verizon.net, comcast.net, ameritech.net. Could indicate spoofing on the internal network or poor network design.

Correlations:

Defensive Recommendations: Implement egress filtering, if traffic that is leaving the LAN does not have a source address of that LAN it is dropped. If external traffic is allowed to cross the internal LAN, the source should be found and eliminated.

Notify Brian B. 3.56 tcp	33 (0%)	20	1
---------------------------------	----------------------------	--------------------	-------------------

Description: A custom rule that detected traffic to my.net.3.56 on ports 21, 53, 80, 666, 3372, 4898 and 4899.

pd95177c7.dip.t-dialin.net 217.81.119.199 --> [my.net.3.56](#)

Analysis:

2003-07-08 03:40:31	195.224.126.6 :4435	my.net.3.56 :53
2003-07-08 06:50:15	217.81.119.199 :26550	my.net.3.56 :666
2003-07-09 00:45:14	212.252.91.20 :35538	my.net.3.56 :21
2003-07-09 00:44:53	212.252.91.20 :35538	my.net.3.56 :21
2003-07-09 00:44:56	212.252.91.20 :35538	my.net.3.56 :21
2003-07-08 06:50:15	217.81.119.199 :21158	my.net.3.56 :4898
2003-07-08 21:41:48	195.223.97.170 :1994	my.net.3.56 :3372
2003-07-09 16:14:03	217.120.249.241 :2039	my.net.3.56 :4899
2003-07-09 16:14:12	217.120.249.241 :2039	my.net.3.56 :4899

The majority of the alerts are on port 80, probably normal traffic. Areas of concern are external traffic to port 666, 53, 21, 4898-99 and 3372. Port 666 has numerous Trojans associated with it: Attack FTP, Back Construction, Cain & Abel, Satanz Backdoor, ServeU, Shadow Phyre, NokNok, BLA Trojan, Th3r1pp3rz [The Rippers]. The other ports are not what you would expect to see external traffic to, unless its part of a scan.

Correlations:

Defensive Recommendations: Check host for compromise. Turn off unused services. Block external connections. This rule appears to be used for logging connections. Snort is a great packet logger, but the output should probably not be included with the other IDS detects.

<u>DDOS shaft client to handler</u> ³⁰	<u>31</u> (0%)	<u>7</u>	<u>5</u>
---	-----------------------------------	--------------------------	--------------------------

Description: Use above hyperlink. This event indicates possible control traffic from the Shaft master to the Shaft handlers.

Analysis:

Traffic to my.net.24.27, my.net.110.80, my.net.108.19 and my.net.6.55 appear to be false positives. The host my.net.my.net appears to be compromised with traffic to it from four external sources.

Correlations:

Defensive Recommendations: Check for compromised host. Block access to external connections.

RFB - Possible WinVNC - 010708-1	<u>30</u> (0%)	<u>15</u>	<u>20</u>
---	-----------------------------------	---------------------------	---------------------------

Description: This is software available from AT&T Labs Cambridge that allows a user to remote control a desktop. Custom rule that appears to alert on a source or destination port of 5900 or 5901.

Analysis:

2003-07-08 14:36:38	<u>68.55.35.44</u> :5900	<u>my.net.114.21</u> :4498	
2003-07-08 14:36:38	<u>my.net.114.21</u> :4498	<u>68.55.35.44</u> :5900	
2003-07-08 16:49:11	<u>my.net.53.128</u> :1229	<u>68.55.35.44</u> :5900	←
2003-07-08 16:49:32	<u>68.55.35.44</u> :5900	<u>my.net.53.128</u> :1232	
2003-07-08 16:49:32	<u>my.net.53.128</u> :1232	<u>68.55.35.44</u> :5900	
2003-07-08 22:21:48	<u>68.55.196.211</u> :20645	<u>my.net.111.51</u> :5900	
2003-07-09 00:31:19	<u>my.net.178.31</u> :5900	<u>151.196.49.115</u> :1207	←
2003-07-09 00:44:38	<u>my.net.111.51</u> :5900	<u>68.55.196.211</u> :26018	
2003-07-09 00:44:38	<u>68.55.196.211</u> :26018	<u>my.net.111.51</u> :5900	
2003-07-09 08:11:36	<u>my.net.111.188</u> :5901	<u>68.55.200.138</u> :18763	
2003-07-09 08:11:36	<u>68.55.200.138</u> :18763	<u>my.net.111.188</u> :5901	
2003-07-09 10:34:24	<u>209.240.190.63</u> :5900	<u>my.net.97.10</u> :1292	
2003-07-09 10:34:24	<u>my.net.97.10</u> :1292	<u>209.240.190.63</u> :5900	
2003-07-09 18:40:52	<u>my.net.111.51</u> :5900	<u>68.55.196.211</u> :33841	
2003-07-09 18:40:52	<u>68.55.196.211</u> :33841	<u>my.net.111.51</u> :5900	
2003-07-09 22:25:09	<u>209.240.190.63</u> :5900	<u>my.net.97.54</u> :1213	←
2003-07-07 15:32:59	<u>209.240.190.63</u> :5900	<u>my.net.97.94</u> :3334	

2003-07-07 15:32:59	my.net.97.94:3334	209.240.190.63:5900	
2003-07-07 20:47:06	209.240.190.63:5900	my.net.98.24:1148	←
2003-07-07 21:36:01	209.240.190.63:5900	my.net.150.203:4534	←
2003-07-06 15:52:56	209.240.190.63:5900	my.net.97.12:1140	←
2003-07-06 22:40:54	209.240.190.63:5900	my.net.98.86:1272	←
2003-07-05 01:20:29	my.net.111.188:5901	68.55.200.138:18613	
2003-07-05 01:20:29	68.55.200.138:18613	my.net.111.188:5901	
2003-07-05 20:48:01	141.156.18.91:5900	my.net.97.46:1123	
2003-07-05 20:48:02	my.net.97.46:1123	141.156.18.91:5900	
2003-07-05 21:39:03	141.156.18.91:5900	my.net.97.46:1198	
2003-07-05 21:39:03	my.net.97.46:1198	141.156.18.91:5900	
2003-07-05 21:42:55	my.net.70.225:5900	68.55.61.117:3647	
2003-07-05 21:42:55	68.55.61.117:3647	my.net.70.225:5900	

A pairing probably indicates a connection between the two hosts. Some traffic was only alerted one way, could indicate an unsuccessful connecti on attempt? Theses hosts should be further investigated for authorized use of WinVNC and/or compromise.

Correlations:

Defensive Recommendations: Check for compromised hosts. Block external access.

Notify Brian B. 3.54 tcp	30 (0%)	19	1
---------------------------------	----------------------------	--------------------	-------------------

Description: A custom rule that detected traffic to my.net.3.54 on ports 21, 53, 80, 3372, 4898 and 4899.

Analysis:

The alerts on port 80 appear to be genuine traffic and are probably false positives. The alerts below were interesting.

2003-07-09 16:14:03 [217.120.249.241:2037](#) [my.net.3.54:4899](#)

2003-07-09 16:14:12 [217.120.249.241:2037](#) [my.net.3.54:4899](#)

This is the same source as the other Brian rule (cc150743-a.assen1.dr.home.nl)

2003-07-08 21:41:45 [195.223.97.170:1992](#) [my.net.3.54:3372](#)

2003-07-08 21:41:48 [195.223.97.170:1992](#) [my.net.3.54:3372](#)

This source (consumer.ta.isynet.it.97.223.195.in-addr.arpa) is also from a foreign country. Not much more can be determined without looking at the packet contents, but given the ports and addresses it is suspicious traffic.

Correlations:

Defensive Recommendations: Check hosts for compromise. Turn off unused services. Block external connections. This rule appears to be used for logging connections. Snort is a great packet logger, but the output should probably not be included with the other IDS detects. Add another Snort box in intrusion detection mode for logging alerts only.

[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	18 (0%)	3	3
--	----------------------------	-------------------	-------------------

Description DCC transfer, Basically this means "Direct Client to Client". Like many other services, IRC allows you to send files to others who are connected. Some channels are

devoted entirely to DCC.

Analysis:

2003-07-08 00:21:32	66.207.164.23 :6667	my.net.80.209 :1036
2003-07-08 00:47:52	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-08 11:47:44	66.207.164.23 :6667	my.net.80.209 :1036
2003-07-08 12:30:29	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-08 12:24:07	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-08 12:29:31	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-08 18:00:44	66.207.164.23 :6667	my.net.80.209 :1036
2003-07-08 20:09:16	66.207.164.23 :6667	my.net.80.209 :1036
2003-07-08 21:10:01	66.207.164.23 :6667	my.net.80.209 :1036
2003-07-09 08:35:01	66.207.164.23 :6669	my.net.80.209 :1986
2003-07-09 17:58:16	216.194.70.11 :7000	my.net.82.36 :4514
2003-07-09 20:47:11	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 11:41:53	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 16:19:40	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 16:30:15	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 16:20:06	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 16:31:50	212.161.35.251 :6667	my.net.74.216 :1026
2003-07-07 18:10:00	66.207.164.23 :6667	my.net.80.209 :1036

IRC clients downloading files from an IRC server.

Correlations:

Defensive Recommendations: Check for compromise. Rebuild a confirmed compromised host. Block external connections. Use good passwords. Turn off file sharing.

TCP SMTP Source Port traffic ³¹		22 (0%)	2	3
Description: Detects source port of 25. This event is generated when possible non-legitimate traffic is detected that should not be allowed through a firewall. This can be used to pass through a poorly configured firewall.				
Analysis:				
2003-07-09 16:54:28	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:28	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:28	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:49	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:49	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:49	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:54:49	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:55:09	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:55:09	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:55:09	64.179.52.39 :25	my.net.25.73 :25		
2003-07-09 16:55:09	64.179.52.39 :25	my.net.25.73 :25		

2003-07-09 16:55:09	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:55:09	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:55:49	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:05	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:05	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:05	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:05	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:07	64.179.52.39:25	my.net.25.73:25
2003-07-09 16:56:07	64.179.52.39:25	my.net.25.73:25
2003-07-07 04:08:52	218.20.215.98:25	my.net.188.147:97
2003-07-06 05:15:57	218.20.215.98:25	my.net.94.5:701

20 of the alerts are from mail.ezinemanager.net 64.179.52.39 to mx8in (my.net.25.73). It is interesting that the source and destination ports are both 25. Need the content of the packets to say anymore otherwise, these are probably normal mail traffic and are false positives. The other two are from the same source to two internal destinations. They need further investigation.

Correlations:

Defensive Recommendations: Check two hosts for possible compromise. Rebuild confirmed compromised hosts. Block external connections. Connections from port 25 should only be allowed to ports >=1024.

[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC ³²	13 (0%)	8	1
Description: Backdoor.Sdbot is a Backdoor Trojan Horse that allows the Trojan's creator to control a computer by using Internet Relay Chat (IRC). Backdoor.Sdbot can update itself by checking for newer versions over the Internet. A family of backdoor Trojans which allow a remote intruder to access and control the computer via IRC channels.			
When run, the Trojan tries to connect to an IRC server and join a specific channel. The Trojan then runs continuously in the background as a server process, listening on the IRC channel for specific commands and carrying out the appropriate actions. When first run, the Trojan may copy itself to the Windows or Windows System folder and create an entry in the registry to run itself on start-up.			
Also known as: IRC-Sdbot [McAfee], Backdoor.IRC.SdBot [KAV], BKDR_SDBOT.B [Trend], Troj/Sdbot-B [Sophos], Win32.SdBot.14176 [CA]			
Analysis:			
2003-07-08 15:31:18	my.net.97.98:2746	213.186.35.9:6667	
2003-07-09 05:14:52	my.net.97.74:1313	213.186.35.9:6667	
2003-07-09 17:16:40	my.net.153.111:4319	213.186.35.9:6667	
2003-07-09 18:09:44	my.net.153.111:4683	213.186.35.9:6667	
2003-07-09 21:31:05	my.net.98.15:1328	213.186.35.9:6667	
2003-07-07 04:55:32	my.net.97.220:3296	213.186.35.9:6667	
2003-07-07 06:47:10	my.net.97.220:3558	213.186.35.9:6667	

2003-07-07 11:15:44	my.net.97.65 :1580	213.186.35.9 :6667
2003-07-07 11:07:27	my.net.97.65 :1441	213.186.35.9 :6667
2003-07-06 00:32:20	my.net.97.84 :1124	213.186.35.9 :6667
2003-07-06 07:04:08	my.net.97.84 :2124	213.186.35.9 :6667
2003-07-06 12:53:49	my.net.97.202 :2376	213.186.35.9 :6667
2003-07-05 23:48:18	my.net.97.84 :1841	213.186.35.9 :6667

The affected hosts are attempting to connect to an IRC server. They appear to be compromised. Most are dialup hosts. They are all connecting to ns336.ovh.net (528 occurrences for this as source) a host in a foreign country.

Correlations:

Defensive Recommendations: Rebuild confirmed compromised hosts. Turn off and remove unneeded services. Enforce a password policy. Configure your email server to block or remove email that contains file attachments that are commonly used to spread viruses, such as .vbs, .bat, .exe, .pif and .scr files. Block external connections.

Traffic from port 53 to port 123 ³³		11 (0%)	1	1
--	--	----------------------------	-------------------	-------------------

Description: One of the things it detects is source port of 53. This event is generated when possible non-legitimate traffic is detected that should not be allowed through a firewall. Traffic from TCP port 53, is used by DNS servers for zone transfers. Normal DNS traffic uses the UDP protocol. An attacker could use a TCP source port of 53 to pass through a poorly configured firewall. DNS traffic from port 53 using either UDP or TCP should be to a port above 1023. Ports 1023 and below are privileged.

Analysis:

2003-07-08 20:40:20	64.125.197.7 :53	my.net.1.3 :123
2003-07-08 23:14:04	64.125.197.7 :53	my.net.1.3 :123
2003-07-09 10:41:13	64.125.197.7 :53	my.net.1.3 :123
2003-07-07 03:02:58	64.125.197.7 :53	my.net.1.3 :123
2003-07-07 04:54:03	64.125.197.7 :53	my.net.1.3 :123
2003-07-07 17:34:05	64.125.197.7 :53	my.net.1.3 :123
2003-07-06 03:42:29	64.125.197.7 :53	my.net.1.3 :123
2003-07-06 12:39:59	64.125.197.7 :53	my.net.1.3 :123
2003-07-06 21:29:20	64.125.197.7 :53	my.net.1.3 :123
2003-07-06 23:21:01	64.125.197.7 :53	my.net.1.3 :123
2003-07-05 22:09:38	64.125.197.7 :53	my.net.1.3 :123

Possible DNS traffic to the nameserver. False Positive.

Correlations:

Defensive Recommendations: Incoming connections from TCP port 53 should only be allowed to machines that need the ability to do zone transfers. Connections from TCP port 53 should only be allowed to ports >=1024 on these machines

Attempted Sun RPC high port access ³⁴		11 (0%)	3	6
--	--	----------------------------	-------------------	-------------------

Description: This event is generated when an attempt is made dump entries from the portmapper on a Solaris host. This request can discover what Remote Procedure Call (RPC) services are offered and on which ports they listen. The portmapper service

registers all RPC services on UNIX hosts. It can be queried for all RPC services running, the RPC program name and version, the protocol (TCP or UDP), and the port where the service listens. This can provide an attacker with valuable information about what RPC services are offered and on which ports. Execute 'rpcinfo -p hostname/IP'.

Analysis:

2003-07-09 05:10:01	63.250.195.10 :49155	my.net.150.121 :32771
2003-07-07 18:26:32	63.250.195.10 :42051	my.net.152.14 :32771
2003-07-07 23:46:20	63.250.195.10 :8301	my.net.150.121 :32771
2003-07-05 20:56:02	63.250.195.10 :21878	my.net.150.203 :32771

The dialups are accessing a dns. These are false positives. The detects with [63.250.195.10](#) as a source are most likely genuine.

Correlations: [63.250.195.10](#) has appeared in many other alerts.

Defensive Recommendations: Check hosts for running service. If running determine if any vulnerabilities on RPC services. Rebuild confirmed compromised hosts. Limit remote access to RPC services. Disable unneeded RPC service.

EXPLOIT NTPDX buffer overflow ³⁵

8
(0%)

4

7

Description: This event is generated when an attempt to exploit a buffer overflow condition in ntpd is made. Some versions of the Network Time Protocol Daemon (ntpd) are vulnerable to a buffer overflow, which can present the attacker with a root shell. Ntp is used to synchronize system time with a time-server. This may also be used on various network devices. Exploit scripts are available

Affected Versions:

ntpd versions prior to and including 4.0.99k
 xntpd and xntp3

Analysis:

2003-07-08 11:50:43	63.250.205.30 :123	my.net.114.110 :123
2003-07-08 14:51:55	63.250.195.10 :19740	my.net.53.49 :123
2003-07-09 04:01:19	63.250.195.10 :123	my.net.150.121 :123
2003-07-09 14:38:33	64.94.235.108 :123	my.net.111.139 :123
2003-07-09 15:05:52	63.250.195.10 :53820	my.net.153.120 :123
2003-07-09 15:05:52	63.250.195.10 :53820	my.net.153.120 :123
2003-07-07 00:36:21	63.250.195.10 :47217	my.net.69.249 :123
2003-07-06 23:24:23	12.129.72.165 :1300	my.net.97.22 :123

[63.250.195.10](#) appears in numerous other alerts. Possible exploited hosts. The systems are being actively targeted. As this signature is specific to an exploit reconnaissance has probably already been done.

Correlations:

Defensive Recommendations: Check hosts to see if service is running. If so check for compromise and rebuild. Upgrade to the latest non-affected version of the software. Apply vendor supplied patches. Block external connections.

NETBIOS NT NULL session ³⁶

8
(0%)

2

5

Description: This event is generated when an attacker sends a blank username and

blank password in an attempt to connect to the IPC\$ (Inter-process Communication) pipe. Information gathering. This attack can permit the disclosure of sensitive information about the target host.

Null sessions allow browsing of Windows hosts by the "Network Neighborhood" and other functions. A Null session permits access to a host using a blank user name and password. An attacker may attempt to perform a Null session connection, disclosing sensitive information about the target host such as available shares and user names.

An attacker can send a blank username and blank password to try to connect to the IPC\$ hidden share on the target computer. Null sessions, may be used by legitimate processes in the same Windows domain.

Analysis:

2003-07-08 15:58:50	200.39.107.90 :4460	my.net.137.37 :139
2003-07-08 15:58:50	200.39.107.90 :4461	my.net.137.34 :139
2003-07-08 16:14:30	200.39.107.90 :1803	my.net.137.36 :139
2003-07-06 05:47:46	212.240.60.65 :2378	my.net.132.45 :139
2003-07-06 05:52:49	212.240.60.65 :3621	my.net.137.34 :139
2003-07-06 05:52:51	212.240.60.65 :3654	my.net.137.46 :139
2003-07-06 05:52:52	212.240.60.65 :3634	my.net.137.37 :139
2003-07-06 05:53:10	212.240.60.65 :3625	my.net.137.36 :139

Traffic from two external hosts in foreign countries connecting to some of the same hosts, check destinations for possible compromise.

Correlations:

Defensive Recommendations: On Windows NT, 2000, XP set the registry key /System/CurrentControlSet/Control/LSA/RestrictAnonymous value to 1. Block external connections. Rebuild confirmed compromised hosts.

DDOS mstream handler to client ³⁷	7 (0%)	1	4
---	---------------------------	-------------------	-------------------

Description: This event is generated when an mstream DDoS handler responds to an mstream client. The mstream DDoS uses a tiered structure of compromised hosts to coordinate and participate in a distributed denial of service attack. At the highest level, clients communicate with handlers to inform them to launch attacks. A client may communicate with a handler using a TCP packet to destination port 12754 with a string of ">" in the payload. A handler responds to this with a TCP source port of 12754 and a string of ">" in the payload.

In late April 2000, we began receiving reports of sites finding a new distributed denial of service (DDOS) tool that is being called "mstream". The purpose of the tool is to enable intruders to utilize multiple Internet connected systems to launch packet-flooding denial of service attacks against one or more target systems. ³⁸

It is important to note that, any of these socket numbers can easily be altered to any value at compile-time by an intruder.

Analysis:

2003-07-08 00:53:43	my.net.6.55 :12754	207.69.200.93 :25
2003-07-08 00:53:43	my.net.6.55 :12754	207.69.200.93 :25
2003-07-08 21:54:59	my.net.6.55 :12754	207.69.200.159 :25
2003-07-09 20:50:49	my.net.6.55 :15104	207.69.200.104 :25
2003-07-09 20:50:49	my.net.6.55 :15104	207.69.200.104 :25
2003-07-06 03:03:04	my.net.6.55 :15104	207.69.200.154 :25
2003-07-06 03:03:04	my.net.6.55 :15104	207.69.200.154 :25
Mail traffic from my.net.6.55 mail server to cave.mail.atl.earthlink.net, james.mail.atl.earthlink.net, carlin.mail.atl.earthlink.net, watson.mail.atl.earthlink.net. Appears to be a false positive.		
Correlations:		
Defensive Recommendations: Tune IDS.		
External FTP to HelpDesk my.net.70.50		6 (0%)
		2
		1
Description: Custom rule that detects traffic to port 21.		
Analysis:		
2003-07-07 20:22:41	195.191.148.212 :1716	my.net.70.50 :21
2003-07-07 20:22:41	195.191.148.212 :1716	my.net.70.50 :21
2003-07-07 20:22:42	195.191.148.212 :1716	my.net.70.50 :21
2003-07-05 00:59:03	62.195.218.52 :3414	my.net.70.50 :21
2003-07-05 00:59:04	62.195.218.52 :3414	my.net.70.50 :21
2003-07-05 00:59:05	62.195.218.52 :3414	my.net.70.50 :21
Two external addresses: node-d-da34.a2000.nl 62.195.218.52 and 195.191.148.212. Appears to be a port scan.		
Correlations: See below rule.		
Defensive Recommendations: Check to see if host is running ftp service. Block external connections.		
External FTP to HelpDesk my.net.70.49		5 (0%)
		2
		1
Description: Custom rule that detects traffic to port 21.		
Analysis:		
2003-07-07 20:22:41	195.191.148.212 :1730	my.net.70.49 :21
2003-07-07 20:22:41	195.191.148.212 :1730	my.net.70.49 :21
2003-07-05 00:59:03	62.195.218.52 :3413	my.net.70.49 :21
2003-07-05 00:59:04	62.195.218.52 :3413	my.net.70.49 :21
2003-07-05 00:59:05	62.195.218.52 :3413	my.net.70.49 :21
Two external addresses: The same as the above External FTP to HelpDesk rule. Occurred at about the same time, with very close source ports. Appears to be a port scan.		
Correlations: See above rule		
Defensive Recommendations: Check to see if host is running the ftp service. Block external connections.		
Probable NMAP fingerprint attempt ³⁹		5
		2
		4

(0%)			
Description: This event is generated when the nmap port scanner and reconnaissance tool is used against a host. When run with the '-O' option, it attempts to identify the remote operating system. Can provide useful reconnaissance information to an attacker. Has been known to cause a denial of service on some older hosts. Analysis: 2003-07-08 14:51:23 141.157.107.131 :0 my.net.12.2 :0 2003-07-08 14:51:23 141.157.107.131 :0 my.net.25.11 :0 2003-07-09 14:42:09 63.251.52.75 :51200 my.net.114.115 :53 2003-07-09 14:42:09 63.251.52.75 :51200 my.net.114.115 :53 2003-07-09 15:36:26 63.251.52.75 :19389 my.net.178.66 :54501 Usually an attacker doing reconnaissance. 63.251.52.75 looks like a false positive. Correlations: Defensive Recommendations: Block external connections.			
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot		4 (0%)	2
Description: Use above hyperlink. Analysis: 2003-07-08 23:30:57 198.163.214.2 :6663 my.net.97.154 :2621 2003-07-09 00:34:06 198.163.214.2 :6662 my.net.97.43 :3415 2003-07-09 01:07:29 198.163.214.2 :6666 my.net.97.43 :3805 2003-07-09 08:38:39 209.221.59.11 :6667 my.net.17.48 :2341 Connections on known server ports. One host makes three connections that are about 30 minutes apart. Probable compromised hosts. Correlations: IP Address: 198.163.214.2 HostName: irc.mpls.ca DShield Profile: Country: CA Contact E-mail: pparrott@PPARROTT.COM AS Number: 0 Total Records against IP: 1826 Number of targets: 810 Date Range: 2003-07-23 to 2003-08-04 Defensive Recommendations: Check for compromise. Rebuild a confirmed compromised host. Block access. Use good passwords. Turn off file sharing.			
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot ⁴⁰		3 (0%)	3
Description: Use above hyperlink. Analysis: 2003-07-08 11:31:44 81.19.252.10 :6667 my.net.17.48 :3615 2003-07-07 12:44:44 216.194.70.9 :7000 my.net.82.41 :1388 2003-07-06 08:06:44 24.94.220.84 :7000 my.net.97.61 :2108 Three hosts connecting to three different destinations. Probably not spoofed. Using			

known Trojan ports. Most likely hosts are compromised and are controlled by IRC.

Correlations:

IP Address: 81.19.252.10
 HostName: ns2.nmnet.dk
 DShield Profile: Country:
 Contact E-mail:
 AS Number: 16095
 Total Records against IP: 244
 Number of targets: 11
 Date Range: 2003-07-23 to 2003-08-16

Defensive Recommendations: Check hosts for compromise. Rebuild a confirmed compromised host. Block external connections.

TFTP - External UDP connection to internal tftp server

3
(0%)

1

2

Description: Custom rule that detects traffic on destination port 69 to internal tftp servers.

Analysis:

2003-07-08 07:39:55 [63.250.195.10](#):50212 [my.net.150.121](#):69
 2003-07-08 13:59:55 [63.250.195.10](#):64434 [my.net.150.121](#):69
 2003-07-09 18:26:34 [63.250.195.10](#):17760 [my.net.152.163](#):69

Same external source (l8.cache.vip.dal.yahoo.com) to two internal tftp servers. Probably not spoofed nor a scan. Compromised host and attacker could be downloading tools. Same source host as Back Orifice detect.

Correlations:

IP Address: 63.250.195.10
 HostName: l8.cache.vip.dal.yahoo.com
 DShield Profile: Country: US
 Contact E-mail: netops@broadcast.com
 AS Number: 5779
 Total Records against IP: 2758
 Number of targets: 202
 Date Range: 2003-07-23 to 2003-08-16

Defensive Recommendations: Possible compromised hosts. Rebuild a confirmed compromised host. Block access from external sources.

[UMBC NIDS IRC Alert] K\line'd user detected, possible trojan.

2
(0%)

2

2

Description: Detects a user being K\lined (see description above for IRC user /kill detected, possible trojan.) possible trojan controlled from IRC.

Analysis:

2003-07-08 15:00:22 [209.221.59.11](#):6667 -> [my.net.17.48](#):2754
 2003-07-08 15:18:15 [208.178.231.190](#):6667 -> [my.net.60.16](#):46190
 Known Trojan port source 6667 Pretty Park, DarkFTP, ScheduleAgent, SubSeven, Subseven 2.14, DefCon 8, Trinity, WinSatan. External traffic to internal hosts on known

Trojan ports. Probably not spoofed.

Correlations: 209.221.59.11 (11 occurrences as source) 208.178.231.190 (10 occurrences as source)

Dshield reports:

IP Address: 208.178.231.190

HostName: irc.Prison.NET

DShield Profile: Country: US

Contact E-mail: ipadmin@gblox.net

AS Number: 3549

Total Records against IP: 86

Number of targets: 31

Date Range: 2003-07-23 to 2003-08-11

Defensive Recommendations: Possible compromised hosts. Rebuild a confirmed compromised host. Check for Trojans. Block access to hosts from external networks.

DDOS mstream client to handler ⁴¹

2
(0%)

2

1

Description: The event is generated when a DDoS mstream client makes contact with an mstream handler. The mstream DDoS uses a tiered structure of compromised hosts to coordinate and participate in a distributed denial of service attack. At the highest level, clients communicate with handlers to inform them to launch attacks. A client may contact a handler using a TCP SYN packet to destination port 15104. After a host becomes an mstream handler, the client will attempt to communicate with the handler

Analysis:

2003-07-08 00:53:43 [207.69.200.93](#):25 [my.net.6.55](#):12754

2003-07-08 21:54:59 [207.69.200.159](#):25 [my.net.6.55](#):12754

Appears to be two mail servers (cave.mail.atl.earthlink.net and james.mail.atl.earthlink.net) contacting the internal mail server my.net.6.55. False positive.

Correlations:

Defensive Recommendations: Tune IDS.

[EXPLOIT FTP passwd retrieval retr path](#) ⁴²

1
(0%)

1

1

Description: This event is generated when an attempt to retrieve a specific file, in this case the systems user database from an FTP server is made. The attacker may obtain a valid list of user names and/or encrypted passwords from the server.

Analysis: One source: ip68-104-181-99.ph.ph.cox.net 68.104.181.99. Six occurrences of this source. Probably not spoofed.

Correlations: 5 other password attempts against my.net.60.11 in 60 seconds, about 72 minutes after this alert.

Defensive Recommendations: Block access from external hosts. Shutdown unnecessary services. Check for compromise.

External FTP to HelpDesk my.net.53.29

1
(0%)

1

1

Description: Custom rule that detects traffic to port 21.

Analysis: One external address: asy20.as91.sol.superonline.com 212.252.91.20. Probably not spoofed. This appears as traffic from a foreign country, trying to transfer files to this host. There are 7 occurrences from this host to internal addresses. They all go to port 21. Judging from the times and source ports it appears to be a scan. Check scans.

Correlations: Dshield reports:

Country: TR

Contact E-mail: solip@superonline.net

AS Number: 6822

Total Records against IP: 3143

Number of targets: 1501

Date Range: 2003-07-08 to 2003-07-08

Defensive Recommendations: Block access from external hosts. Shutdown unnecessary services.

[Back Orifice](#) ⁴³

[1](#)
(0%)

[1](#)

[1](#)

Description: BackOrifice is a Trojan Horse consisting of two main pieces, a client application and a server application. The client application, running on one machine, can be used to monitor and control a second machine running the server application.

Analysis: Although this rule could have a high incident of false positives, this looks like actual BO activity, the source port is a high port and the destination port is 31337. The source IP of this activity has 80 other occurrences. Probably not spoofed.

Correlations: One source: l8.cache.vip.dal.yahoo.com 63.250.195.10 There are 81 occurrences of various activity from this source. MyNetWatchman reports 10011 events, from 23 agents for this source.

Defensive Recommendations: Possible compromised host. Check host for listening port 31337. Check host for startup of BO server application. Rebuild a confirmed compromised host. Block access from external hosts.

CS WEBSERVER - external ssh traffic

[1](#)
(0%)

[1](#)

[1](#)

Description: A custom rule to detect traffic to port 22.

Analysis: One external address: 213.35.200.178 213-35-200-178-dsl.kvm.estpak.ee. This traffic is probably not spoofed.

Correlations: Dshield reports 4 records against this IP. There are two other occurrences with this source. Ssh to my.net.30.3 and my.net.30.4.

Appears to be a scan judging by the timestamps and source ports.

Defensive Recommendations: Block external ssh to these internal hosts.

Top Talkers

Alert Log Analyses

ACID

Unique Source Address(es): 15 Most Frequent IP addresses

[Home](#)
[Search](#) | [AG Maintenance](#)

Queried DB on : Fri July 25, 2003 09:12:42

Displaying 15 Most Frequent IP addresses

≤ Src IP address ≥	Sensor #	≤ Total # ≥	≤ Unique Alerts ≥	≤ Dest. Addr. ≥
my.net.153.185	1	19413	1	39
65.214.36.116	1	11715	3	3
my.net.198.221	1	5864	2	1
169.254.45.176	1	4587	2	108
24.35.42.249	1	3499	1	1
my.net.97.168	1	3469	1	1
my.net.97.38	1	3172	2	6
my.net.162.41	1	3057	1	1
my.net.111.34	1	2353	3	4
63.164.243.132	1	1576	1	1
68.55.226.150	1	1561	2	2
my.net.97.29	1	1316	1	6
my.net.97.243	1	1251	1	24
213.204.59.157	1	1202	1	1
211.114.9.211	1	1096	1	1044

[Loaded in 9 seconds]

ACID v0.9.6b23 (by [Roman Danyliw](#) as part of the [AirCERT](#) project)

Out of Spec Analyses

ACID	Unique Source Address(es): 15 Most Frequent IP addresses	Home	
		Search	AG Maintenance

Queried DB on : Thu July 24, 2003 12:10:04

Displaying 15 Most Frequent IP addresses

≤ Src IP address ≥	Sensor #	≤ Total # >	≤ Unique Alerts ≥	≤ Dest. Addr. >
142.26.120.7	1	24758	1	24758
194.238.50.12	1	1317	1	6
213.186.35.9	1	780	1	14
67.119.233.217	1	579	1	5
80.143.95.179	1	418	1	1
80.143.121.205	1	415	1	2
216.95.201.21	1	371	1	11
216.95.201.29	1	365	1	7
168.226.117.108	1	331	1	1
216.95.201.25	1	331	1	10
216.95.201.22	1	329	1	10
216.95.201.28	1	303	1	10
216.95.201.24	1	284	1	8
216.95.201.27	1	282	1	8
216.95.201.23	1	275	1	8

[Loaded in 1 seconds]

ACID v0.9.6b23 (by [Roman Danyliw](#) as part of the [AirCERT](#) project)

External Source Addresses and Registration Information

For registration info I chose the top out-of-spec talker 142.26.120.7, probably doing a lot of reconnaissance. The top attacker IP address of 63.250.195.10. The my.net.30.4 activity source IP address of 217.120.249.241 . Possible Trojan activity source IP address of 80.218.101.63. Exploit x86 NOOP source IP address of 217.106.116.202.

Top out-of-spec talker 142.26.120.7

OrgName: British Columbia Systems Corporation
OrgID: BCSC
Address: 400 Seymour Place
City: Victoria
StateProv: BC
PostalCode: V8X-4S8
Country: CA

NetRange: 142.26.0.0 - 142.26.255.255
CIDR: 142.26.0.0/16
NetName: BCSYSTEMS5
NetHandle: NET-142-26-0-0-1
Parent: NET-142-0-0-0-0
NetType: Direct Assignment
NameServer: DNS.GOV.BC.CA
NameServer: DNS1.GOV.BC.CA
NameServer: DNS2.GOV.BC.CA
NameServer: DNS3.GOV.BC.CA
Comment:
RegDate: 1991-05-13
Updated: 1998-09-16

TechHandle: AT110-ARIN
TechName: Teasdale, Alan
TechPhone: +1-250-387-5577
TechEmail: al.teasdale@gems2.gov.bc.ca

OrgAbuseHandle: CSC28-ARIN
OrgAbuseName: Customer Service Centre
OrgAbusePhone: +1-250-952-6000
OrgAbuseEmail: cshelp@gems3.gov.bc.ca

OrgNOCHandle: CSC28-ARIN
OrgNOCName: Customer Service Centre
OrgNOCPhone: +1-250-952-6000
OrgNOCEmail: cshelp@gems3.gov.bc.ca

OrgTechHandle: AT110-ARIN
OrgTechName: Teasdale, Alan
OrgTechPhone: +1-250-387-5577
OrgTechEmail: al.teasdale@gems2.gov.bc.ca

Top attacker IP address of 63.250.195.10

OrgName: Yahoo! Broadcast Services, Inc.
OrgID: YAHO
Address: 701 First Avenue
City: Sunnyvale
StateProv: CA
PostalCode: 94089
Country: US

NetRange: 63.250.192.0 - 63.250.223.255
CIDR: 63.250.192.0/19
NetName: NETBLK2-YAHO OBS
NetHandle: NET-63-250-192-0-1
Parent: NET-63-0-0-0-0
NetType: Direct Allocation
NameServer: NS1.YAHOO.COM
NameServer: NS2.YAHOO.COM
NameServer: NS3.YAHOO.COM
NameServer: NS4.YAHOO.COM
NameServer: NS5.YAHOO.COM
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate: 1999-11-24
Updated: 2003-05-06

TechHandle: NA258-ARIN
TechName: Netblock Admin, Netblock
TechPhone: +1-408-349-7183
TechEmail: netblockadmin@yahoo-inc.com

The my.net.30.4 activity source IP address of 217.120.249.241 .

Final results obtained from whois.ripe.net.

Results:

% This is the RIPE Whois server.

% The objects are in RPSL format.

%

% Rights restricted by copyright.

% See <http://www.ripe.net/ripenc/db/copyright.html>

inetnum: 217.120.248.0 - 217.120.249.255

netname: AT HOME-BENELUX-ASSEN-8

descr: @Home Benelux Assen Headend block

country: NL

```
admin-c: ABNO1-RIPE
tech-c: HOME2-RIPE
remarks: Please report abuse by email to abuse@home.nl
remarks: INFRA-AW
status: ASSIGNED PA
mnt-by: BENELUX-MNT
mnt-lower: BENELUX-MNT
changed: ahulsebos@corp.home.nl 20030321
source: RIPE

route: 217.120.0.0/14
descr: @Home Benelux
origin: AS9143
remarks: -----
remarks: E-mail is the preferred contact method!
remarks: -----
remarks: Please use one of the following addresses:
remarks: abuse@home.nl - for abuse notifications
remarks: ripe-athome@corp.home.nl - for technical questions
remarks: -----
mnt-by: IPMGMT-RIPE
changed: andre@corp.home.nl 20020920
source: RIPE

role: AtHome Benelux Network Operations Centre
address: Gyroscopweg 90-92
address: 1042 AX Amsterdam
phone: +31 20 885 5544
fax-no: +31 20 885 5525
e-mail: noc@corp.home.nl
trouble: Please report abuse by e-mail to abuse@home.nl
admin-c: AVL52-RIPE
tech-c: HOME2-RIPE
nic-hdl: ABNO1-RIPE
notify: ripe-athome@corp.home.nl
mnt-by: IPMGMT-RIPE
changed: andre@corp.home.nl 20020920
source: RIPE

role: AtHome Benelux IP Management
address: Gyroscopweg 90-92
address: 1042 AX Amsterdam
address: The Netherlands
phone: +31 20 885 5588
fax-no: +31 20 885 5525
e-mail: ripe-athome@corp.home.nl
```

```
trouble: Report abuse by e-mail to abuse@home.nl
admin-c: AVL52-RIPE
tech-c: AVL52-RIPE
tech-c: AH218-RIPE
tech-c: JVV19-RIPE
tech-c: BOR7-RIPE
tech-c: HI62-RIPE
tech-c: KM754-RIPE
nic-hdl: HOME2-RIPE
notify: ripe-athome@corp.home.nl
changed: andre@corp.home.nl 20030329
source: RIPE
```

Possible Trojan activity source IP address of 80.218.101.63

Final results obtained from whois.ripe.net.

Results:

% This is the RIPE Whois server.

% The objects are in RPSL format.

%

% Rights restricted by copyright.

% See <http://www.ripe.net/ripenc/pdb-services/db/copyright.html>

inetnum: 80.218.0.0 - 80.218.107.255

netname: CABLECOM-MAIN-NET

descr: Cablecom GmbH

descr: Zuerich

country: CH

remarks: *****

remarks: For spam/abuse, please contact abuse@cablecom.ch

remarks: E-mails to the persons below will be IGNORED!!

remarks: *****

remarks: INFRA-AW

admin-c: WM5132-RIPE

tech-c: CAN6-RIPE

status: ASSIGNED PA

notify: lir-mnt@cablecom.ch

mnt-by: AS8404-MNT

changed: wilson.mehringer@cablecom.ch 20021212

source: RIPE

route: 80.218.0.0/15

descr: Cablecom GmbH

```
descr: Zollstrasse42
descr: CH-8021 Zuerich
descr: SWITZERLAND
origin: AS8404
remarks: *****
remarks: For Spam/Abuse, please contact abuse@cabecom.ch
remarks: E-mails to the persons below will be IGNORED!!
remarks: *****
notify: lir-mnt@cabecom.ch
mnt-by: AS8404-MNT
changed: felix.giger@cabecom.ch 20020524
changed: wilson.mehringer@cabecom.ch 20020530
source: RIPE

role: Cablecom GmbH NOC
address: Zollstrasse 42
address: CH-8021 Zuerich
remarks: *****
remarks: For spam/abuse, please contact abuse@cabecom.ch
remarks: E-mails to the persons below will be IGNORED!!
remarks: *****
fax-no: +41 1 277 93 22
e-mail: wilson.mehringer@cabecom.ch
admin-c: WM5132-RIPE
tech-c: PG4227-RIPE
nic-hdl: CAN6-RIPE
notify: lir-mnt@cabecom.ch
mnt-by: AS8404-MNT
changed: pascal.gruenig@cabecom.ch 20000814
changed: wilson.mehringer@cabecom.ch 20011129
changed: wilson.mehringer@cabecom.ch 20020124
source: RIPE

person: Wilson Mehringer
address: Cablecom GmbH
address: Zollstrasse 42
address: CH-8021 Zurich
address: Switzerland
phone: +41 1 277 90 72
remarks: *****
remarks: For Spam/Abuse, please contact abuse@cabecom.ch
remarks: E-mails to the persons below will be IGNORED!!
remarks: *****
e-mail: wilson.mehringer@cabecom.ch
nic-hdl: WM5132-RIPE
notify: wilson.mehringer@cabecom.ch
```



```
mnt-by: AS8404-MNT
changed: wilson.mehringer@cablecom.ch 20020130
changed: wilson.mehringer@cablecom.ch 20020808
changed: wilson.mehringer@cablecom.ch 20021107
source: RIPE
```

Exploit x86 NOOP source IP address of 217.106.116.202.

Final results obtained from whois.ripe.net.

Results:

% This is the RIPE Whois server.

% The objects are in RPSL format.

%

% Rights restricted by copyright.

% See <http://www.ripe.net/ripenncc/pub-services/db/copyright.html>

inetnum: 217.106.116.0 - 217.106.117.255

netname: VORONEJ-RU1

descr: Comincom-Voronej

country: RU

admin-c: SY252-RIPE

admin-c: OS251-RIPE

tech-c: SY252-RIPE

tech-c: OS251-RIPE

status: ASSIGNED PA

notify: sow@comch.ru

notify: registry@rt.ru

mnt-by: AS8342-MNT

changed: rus@rt.ru 20030121

source: RIPE

route: 217.106.0.0/15

descr: ROSTELECOM-NET

origin: AS8342

notify: ncc@rt.ru

mnt-by: AS8342-MNT

changed: rus@rt.ru 20001221

source: RIPE

person: Sergey Yakimenko

address: Comincom Chernozemiye Joint Stock Company

address: Lenina sq.12

address: 394000 Voronezh

address: Russia
phone: +7 0732 521233
phone: +7 0732 776828
phone: +7 502 2002008
fax-no: +7 0732 521233
e-mail: ysu@comch.ru
nic-hdl: SY252-RIPE
changed: sow@comch.ru 19971006
source: RIPE

person: Oleg Semenihih
address: Comincom Chernozemiye Joint Stock Company
address:
address: 394000 Voronezh
address: Russia
phone: +7 0732 552679
fax-no: +7 0732 521233
e-mail: sow@comch.ru
nic-hdl: OS251-RIPE
mnt-by: DENIC-P
changed: sow@comch.ru 19971006
changed: hostmaster@denic.de 20000621
source: RIPE

© SANS Institute 2004

References

-
- ¹ Martin Roesch. "Snort: Packet Logger Intrusion Detection System"
URL: <http://www.snort.org/>
- ² Jason Haar. Logsnorter-0.2.
URL: http://www.snort.org/dl/contrib/other_logs/logsnorter-0.2.tar.gz
- ³ Roman Danyliw. The Analysis Console for Intrusion Databases.
URL: <http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html>
- ⁴ C0ldPhaTe. "Microsoft IIS Unicode Exploit Explained." URL:
<http://www.astalavista.com/library/os/iis/>
- ⁵ Jim Forster. Subject: Re: [snort] 'SMB Name Wildcard' Date: Mon Jan 17 2000 - 09:26:14 CST URL: <http://archives.neohapsis.com/archives/snort/2000-01/0222.html>
- ⁶ TonikGin. "XDCC – An .EDU Admin's Nightmare. Sept. 11 2002."
URL: <http://www.russonline.net/tonikgin/eduhacking.html>
- ⁷ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center." "Advisory CA-2001-23 Continued Threat of the "Code Red" Worm.
URL: <http://www.cert.org/advisories/CA-2001-23.html>
- ⁸ Gunther Birznies. "Web Application Security."
URL: http://www.extropia.com/presentations/birznies/pdf/cgi_security_history.pdf
- ⁹ Jon Hart. Snort Signature Database. "Documentation for rule SHELLCODE x86 NOOP."
URL: <http://www.snort.org/snort-db/sid.html?sid=648>
- ¹⁰ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center." "CERT® Advisory CA-2003-10 Integer overflow in Sun RPC XDR library routines."
URL: <http://www.cert.org/advisories/CA-2003-10.html>
- ¹¹ Global Incident Analysis Center. Description of Ramen Worm.
URL: <http://www.sans.org/y2k/ramen.htm>
- ¹² F-Secure Security Information Center. "Virus descriptions Code Red."
URL: <http://www.europe.f-secure.com/v-descs/bady.shtml>

- ¹³ Vivek Sharma. "Scan of the Month Challenge - Scan 23."
URL: <http://honeynet.hackers.nl/scans/scan23/sol/Vivek.html>
- ¹⁴ Jon Hart. Snort Signature Database. "Documentation for rule SCAN NULL."
URL: <http://www.snort.org/snort-db/sid.html?sid=623>
- ¹⁵ LinuxSecurity.com Features. "Scanning and Defending Networks with Nmap."
URL: http://www.linuxsecurity.com/feature_stories/feature_story-4.html
- ¹⁶ The MITRE Corporation. "Common Vulnerabilities and Exposures (CVE)."
CVE-2000-0917. Jan. 2001. URL: <http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=lpng>
- ¹⁷ Netscape Chat. "Supported IRC User-based Query Commands."
URL: <http://wp.netscape.com/eng/chat/2.0/handbook/00000106.htm>
- ¹⁸ ONCtek Ltd. "List of known Trojan/Backdoors."
URL: <http://www.onctek.com/trojanports.html>
- ¹⁹ Christopher Heng. "Nimbdaworm / Virus: What Are Cmd.Exe, Readme.Eml, Readme.Exe, Root.Exe ?." 26 September 2001.
URL: <http://www.thesitewizard.com/news/nimbdaworm.shtml>
- ²⁰ Nigel Houghton. Josh Sakofsky. Snort Signature Database. "Documentation for rule NETBIOS SMB C\$ access." URL: <http://www.snort.org/snort-db/sid.html?sid=533>
- ²¹ Matt Kettler. Mar 19 2003. "Re: [Snort-users] SNMP public access udp."
URL: <http://archives.neohapsis.com/archives/snort/2003-03/0792.html>
- ²² Brian Caswell. Nigel Houghton. Chaos. Snort Signature Database. "Documentation for rule SNMP public access udp."
URL: <http://www.snort.org/snort-db/sid.html?sid=1411>
- ²³ Dragos Ruiu. Feb 12 2001. "In reply to Graham Bevan."
URL: <http://archives.neohapsis.com/archives/snort/2001-02/0320.html>
- ²⁴ Duke University. "OIT Security." May 2002
URL: <http://www.oit.duke.edu/security/cleaning/xdcc.html>
- ²⁵ Jon Hart. Snort Signature Database. "Documentation for rule SHELLCODE x86 setuid 0." URL: <http://www.snort.org/snort-db/sid.html?sid=650>

²⁶ Matt Kettler. Snort Signature Database. “Documentation for rule SHELLCODE x86 stealth NOOP.” URL: <http://www.snort.org/snort-db/sid.html?sid=651>

²⁷ Max Vision. Anton Chuvakin. Nigel Houghton. Snort Signature Database. “Documentation for rule FTP passwd retrieval attempt.” URL: <http://www.snort.org/snort-db/sid.html?sid=356>

²⁸ Nigel Houghton. Nick Black. Snort Signature Database. “Documentation for rule MISC Tiny Fragments.” URL: <http://www.snort.org/snort-db/sid.html?sid=522>

²⁹ Arachnids - The Intrusion Event Database. “Documentation for rule IDS284/SHELLCODE_SHELLCODE-X86-SETGID0.” URL: <http://www.digitaltrust.it/arachnids/IDS284/event.html>

³⁰ Arachnids The Intrusion Event Database. “Documentation for rule IDS254/DDOS_DDOS-SHAFT-CLIENT-TO-HANDLER.” URL: <http://www.digitaltrust.it/arachnids/IDS254/event.html>

³¹ Nigel Houghton. Steven Alexander. Snort Signature Database. “Documentation for rule MISC Source Port 20 to <1024.” URL: <http://www.snort.org/snort-db/sid.html?sid=503>

³² Symantec Security Response. Backdoor SDbot URL: <http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html>

³³ Nigel Houghton. Steven Alexander. Snort Signature Database. “Documentation for rule MISC source port 53 to <1024.” URL: <http://www.snort.org/snort-db/sid.html?sid=504>

³⁴ Max Vision. Brian Caswell. Judy Novak. Snort Signature Database. “Documentation for rule RPC portmap listing TCP 32771.” URL: <http://www.snort.org/snort-db/sid.html?sid=599>

³⁵ Nigel Houghton. Snort Signature Database. “Documentation for rule EXPLOIT ntpdx overflow attempt.” URL: <http://www.snort.org/snort-db/sid.html?sid=312>

³⁶ Steven Alexander. Nawapong Nakjang. Judy Novak. Snort Signature Database. “Documentation for rule NETBIOS NT NULL session.” URL: <http://www.snort.org/snort-db/sid.html?sid=530>

³⁷ Judy Novak. Snort Signature Database. “Documentation for rule DDOS mstream handler to client .” URL: <http://www.snort.org/snort-db/sid.html?sid=248>

³⁸ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center." Incident Note IN-2000-05. "mstream Distributed Denial of Service Tool." May 2000. URL: http://www.cert.org/incident_notes/IN-2000-05.html

³⁹ Nigel Houghton. Steven Alexander. Snort Signature Database. "Documentation for rule SCAN nmap fingerprint attempt." URL: <http://www.snort.org/snort-db/sid.html?sid=629>

⁴⁰ Duke University. "OIT Security." May 2002
URL: <http://www.oit.duke.edu/security/cleaning/xdcc.html>

⁴¹ Judy Novak. Snort Signature Database. "Documentation for rule DDOS mstream client to handler." URL: <http://www.snort.org/snort-db/sid.html?sid=249>

⁴² Max Vision. Anton Chuvakin. Nigel Houghton. Snort Signature Database. "Documentation for rule FTP passwd retrieval attempt." URL: <http://www.snort.org/snort-db/sid.html?sid=356>

⁴³ Symantec Security Updates. Information on Back Orifice and NetBus
URL: <http://www.symantec.com/avcenter/warn/backorifice.html>

Appendix

Alert Descriptions

spp_http_decode: IIS Unicode attack detected¹

Microsoft Internet Information Server (IIS) versions 4.0 and 5.0 which usually runs on Windows NT4 and Windows 2k all have the Unicode extensions installed by default. Unicode allows characters that are not used in the English language to be recognized by Web Servers. The Unicode IIS Exploit allows users to run arbitrary commands on the target web servers. The Unicode extensions loaded on IIS Servers are known to be vulnerable unless they are running the current patches within the server. The Unicode exploit uses Unicode representation of a directory delimiter (/) to fool IIS.

CVE-2000-0884² IIS 4.0 and 5.0 allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain UNICODE encoded characters, aka the "Web Server Folder Traversal" vulnerability.

User access should be restricted to an assigned web root directory and subdirectories when interacting with a web server. Attackers who attempt to perform directory traversals outside the web root should be denied access. A vulnerability exists in IIS web servers that allows directory traversal outside the web root directory when Microsoft double encoding of specific characters is used. This particular attack uses the double encoding of the "/" to escape the web root. This may permit an attacker to execute commands on the vulnerable server.³

SMB Name Wildcard⁴

SMB Wildcard a generic search (wildcard) to query a host for its NetBIOS table. This signature was created and can be reproduced by using the unix samba command "nmblookup -A ". By accessing system name table information, individuals can obtain information which can be used to launch an attack. Information available includes: 1. The NetBIOS name of the server. 2. The Windows NT workgroup domain name. 3. Login names of users who are logged into the server. 4. The name of the administrator account if they are logged into the server. It is considered best practice to ensure that users outside of your network are not permitted to access the NetBIOS name service. This is usually accomplished by configuring packet filters to drop UDP traffic to port 137.

PACKET TRACES 12/30-02:28:32.282973 source:1057 -> target:137
UDP TTL:64 TOS:0x0 ID:62089

Len: 58

```
24 C0 00 00 00 01 00 00 00 00 00 00 20 43 4B 41 $..... CKA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 00 00 21 AAAAAAAAAAAAAAA..!
00 01
```

[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC⁵

Intro to IRC

IRC is a worldwide network of computers all setup for one purpose, communication. People can come to IRC to chat with friends or meet new people, discuss hot topics such as politics, religion, or breaking news. Over the recent years it has gained much fame, much due to popularity in the warez scene. Warez, simply defined, is the illegal downloading of copyrighted material. Groups which have access to pre-released games, are willing to sneak a camera into a theatre, or happened to beta test the newest Microsoft OS, are eagerly willing to digitize these formats, and make them readily available on the internet for the masses. How does IRC fit into this? IRC is one meeting place people (deemed leechers) can come to congregate and download these files.

B) Intro to File Sharing

Ahhh.. the wonders of connecting to a server, finding a file, and downloading it. Sure is easier than going to Best Buy and buying the game (and usually quicker). So, what is exactly file sharing? Simply... sharing files. Large amounts of people connect to servers, where they are all 'connected' to each other, to download files off others hard drives. IRC has a file server feature, where people can connect, view files on your machine, and download whatever you give them access too. But there are also services such as Kazza, BearShare, Napster, LimeWire, and many more which when you search for a file, your looking through everyones computer at once. That is what it is all about. How is file sharing related to this article? Read on...

C) Intro to XDCC

Pay attention, this is where things pick up. XDCC revolutionized IRC. Many people now use IRC because of this new 'XDCC' feature. What is it? Like a file server, yet automated. It will periodically list the files (usually 1-5 large files) in the channel (chat room) which it is hosting, for people to download. There is a program called Iroffer (1) which makes this even easier. It will (using the definitions in a configuration file you setup), connect to an IRC server, join a channel, and automatically list files.

High port 65535 tcp - possible Red Worm – traffic ⁶

CERT[®] Advisory CA-2001-23 Continued Threat of the "Code Red" Worm

The "Code Red" worm is malicious self-propagating code that exploits Microsoft Internet Information Server (IIS)-enabled systems susceptible to the vulnerability described in [CA-2001-13 Buffer Overflow In IIS Indexing Service DLL](#). Its activity on a compromised machine is time sensitive; different activity occurs based on the date (day of the month) of the system clock. The CERT/CC is aware of at least two major variants of the worm, each of which exhibits the following pattern of behavior:

Propagation mode (from the 1st - 19th of the month): The infected host will attempt to connect to TCP port 80 of randomly chosen IP addresses in order to further propagate the worm. Depending on the configuration of the host that receives this request, there are varied consequences. Unpatched, IIS 4.0 and 5.0 servers with Indexing service installed will almost certainly be compromised by the "Code Red" worm. In the earlier variant of the worm, victim hosts with a default language of English experienced a defacement on all pages requested from the web server. Hosts infected with the later variant did not experience any change in the served content.

spp_http_decode: CGI Null Byte attack detected ⁷

Perl's open command passes the filename to the OS. The problem is that the OS system calls treat null bytes (\0) as ending the string. But Perl does not. So if we have a url like: `http://x.com/cgi-in/vulnerable.cgi?file=/etc/passwd%00.dat` the file passes the Perl regex because it does end in .dat but the system call to open the file ends in /etc/passwd allowing the user access.

In Snort it's (newly) part of the http preprocessor. Basically, if the http decoding routine finds a %00 in an http request, it will alert with this message. Sometimes you may see false positives with sites that use cookies with URL encoded binary data, or if you're scanning port 443 and picking up SSL encrypted traffic. If you're logging alerted packets you can check the actual string that caused the alert. Also, the Unicode alert is subject to the same false positives with cookies and SSL. Having the packet dumps is the only way to tell for sure if you have a real attack on your hands, but this is true for any content-based alert.

-Joe ⁸

These messages are produced by the http_decode preprocessor. If you wish to turn these checks off, add -unicode or -cginull to your http_decode preprocessor line respectively. `preprocessor http_decode: 80 8080 -unicode -cginull`

EXPLOIT x86 NOOP ⁹

A series of NOP instructions for Intel's x86 architecture was detected. As part of an attack on a remote service, an attacker may attempt to take advantage of insecure coding practices in hopes of executing arbitrary code. This procedure

generally makes use of NOPs. The NOP allows an attacker to fill an address space with a large number of NOPs followed by his or her code of choice. This allows "sledding" into the attackers shellcode.

If a particular service was written using unsafe functions without bounds checking (`strcpy()`, `strcat()`, `sprintf()` etc...), it is possible to write arbitrary data to the address space of the service. Normally, this may just cause the program to die a horrible death. However, if you can get the return address to point to the beginning of the newly written data, it is possible to execute code of your choice. This requires that the newly written data is actual executable data. Since calculating exactly where the return address may point to, is no small task, a popular technique is to pad the space leading up to your shellcode with NOPs. This way, if the return address, points anywhere in the series of NOPS, execution will slide down into your shellcode.

External RPC call ¹⁰

Sun RPC commonly has a portmapper listening on port 111, and RPC services of various sorts listening on ports from 32771-34000. These services are often exploitable.

XDR (external data representation) libraries are used to provide platform-independent methods for sending data from one system process to another, typically over a network connection. Such routines are commonly used in remote procedure call ([RPC](#)) implementations to provide transparency to application programmers who need to use common interfaces to interact with many different types of systems.

The `xdrmem_getbytes()` function in the XDR library provided by Sun Microsystems contains an [integer overflow](#) that can lead to improperly sized dynamic memory allocation. Depending on how and where the vulnerable `xdrmem_getbytes()` function is used, subsequent problems like buffer overflows may result.

By calling the portmapper an attacker can determine which services are running and what ports they are listening on.¹¹ Execute 'rpcinfo -p hostname/IP'.

IDS552/web-iis_IIS ISAPI Overflow ida nosize

L-098: Microsoft Index Server ISAPI Extension Buffer Overflow ¹²

As part of its installation process, IIS installs several ISAPI extensions -- .dlls that provide extended functionality. Among these is `idq.dll`, which is a component of Index Server (known in Windows 2000 as Indexing Service) and provides support for administrative scripts (.ida files) and Internet Data Queries (.idq files).

A security vulnerability results because idq.dll contains an unchecked buffer in a section of code that handles input URLs. An attacker who could establish a web session with a server on which idq.dll is installed could conduct a buffer overrun attack and execute code on the web server. Idq.dll runs in the System context, so exploiting the vulnerability would give the attacker complete control of the server and allow him to take any desired action on it.

The worm exploits the Index Server (.ida) buffer overflow vulnerability reported in CIAC Bulletin L-098, *Microsoft Index Server ISAPI Extension Buffer Overflow* and Microsoft Security Bulletin MS01-033, *Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise*. The buffer overflow allows the worm to execute code within the IIS server to spread itself, to deface the server's home page, and to run a denial of service attack on www.whitehouse.gov.

The worm arrives at the web server as a Get /default.ida request. That request exploits the .ida vulnerability and starts the worm code executing. The worm code executes only in memory so no residue of the worm will be found by examining the disk.

NMAP TCP ping! ¹³

A TCP "ping" will send an ACK to each machine on a target network. Machines that are up should respond with a TCP RST. To use the TCP "ping" option with a ping scan, include the "-PT" flag to target a specific port on the network you're probing. In our example, we'll use port 80 (http), which is the default, and it will probably be allowed through the target's border routers and possibly even its firewall. Note that the targeted port does not need to be open on the hosts that are being probed to determine if the machine is up or not. Launch this type of scan as follows:

```
# nmap -sP -PT80 192.168.7.0/24
TCP probe port is 80
```

```
Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
Host (192.168.7.11) appears to be up.
Host (192.168.7.12) appears to be up.
Host (192.168.7.76) appears to be up.
Nmap run completed -- 256 IP addresses (3 hosts up) scanned in 1 second
```

The signature is usually an ACK with a sequence number of zero.

SMB C access ¹⁴

Windows hosts have a default administrative share of the local hard drives. Using the format %DRIVE_LETTER% + \$. Anybody with administrative rights

can remotely access the share. An attacker may be attempting to access files located on the C drive of the host.

Intruders are actively exploiting Windows networking shares that are made available for remote connections across the Internet. This is not a new problem, but the potential impact on the overall security of the Internet is increasing.¹⁵

Unprotected, Windows networking shares can be exploited by intruders in an automated way to place tools on large numbers of Windows-based computers attached to the Internet. Because site security on the Internet is interdependent, a compromised system not only creates problems for the system's owner, but it is also threat to other sites on the Internet. The greater immediate risk to the Internet community is the potentially large number of systems attached to the Internet with unprotected Windows networking shares combined with distributed attack tools.

EXPLOIT x86 setuid 0¹⁶

Shellcode to set the user identity to 0 (root) was detected. If this code is executed successfully, it is possible for the current process to inherit root privileges. However, setuid (2) requires root privileges to be executed in the first place, if the current UID is attempting to get a higher privilege level.

As part of an attack on a remote service, an attacker may attempt to take advantage of insecure coding practices and execute code of his or her choosing through techniques known as 'buffer-overflows', 'format-strings' and others. Such attacks may contain code to change the identity of the current user to that of the root account (setuid0).

Large binary transfers, certain web traffic, and even mail traffic can trigger this rule, but are not necessarily indicative of actually setuid code. Determine what stream of traffic generated this particular alert. If you only have the alert but not the entire packet, examine system for peculiarities. If you are smart and have the entire packet (or better yet, all your traffic for the past n hours), attempt to determine if this particular sequence of characters was part of an innocent stream of data (large binary transfers, for example) or part of a malicious act against your machine. In either case, check for other activity from the host in question -- both currently collected traffic and traffic in the future.

EXPLOIT x86 stealth noop¹⁷

Binary data in the packet matched one kind of byte sequence used as filler in buffer overflow attacks. It is possible someone was attempting a buffer overflow to gain unauthorized access to one of your servers. This rule triggers when a binary pattern appears in the packet contents, which matches one form of filler-

bytes used in buffer overflow attacks. Buffer overflows allow execution of arbitrary code with the privilege level of the affected server process.

A very detailed discussion of how basic buffer overflows work can be found in the text of, "Smashing the stack for fun and profit", by Aleph One in Phrack #49. If the attacker suspects you have a server, which is vulnerable to buffer overflow, they will attempt to exploit this vulnerability to gain access. Tools that use buffer overflows with stealth nop are widely available.

This byte pattern can naturally occur in almost any binary data, so file downloads, streaming media, etc can cause this to false positive. If this traffic appears to be coming from a web or ftp server outside your network to one of your client machines, it is likely a false alert caused by someone downloading a binary file. If this was directed at a port on one of your machines, which is running a server process, you may want to check to see if it has been exploited.

FTP passwd attempt¹⁸

This event is generated when an attempt to retrieve a specific file, in this case the systems user database from an FTP server is made. The attacker may obtain a valid list of user names and/or encrypted passwords from the server. This event is generated when an attempt to download a copy of the "passwd" file from the server is made. The UNIX "passwd" file (typically located in "/etc/" directory) is used to hold the authentication information for system logins. This file needs to be readable by all system users.

Where shadow passwords are used, the actual encrypted passwords are stored in a separate file, only readable by root. It is possible to use password-cracking tools, to obtain unencrypted passwords, either by trying random character combinations, a predefined word list or a combination of public user information. The attacker may use the information contained in the passwd file to launch a dictionary attack against the victim host or other hosts the same users may have access to.

The attacker downloads a "passwd" file from a machine that does not use shadowed passwords and uses a tool like John-the-Ripper to crack the passwords used for several accounts. He then proceeds to login to the system remotely and possibly gain escalated privileges via a local exploit on the system.

The attack usually requires FTP access to the /etc/ directory either by system mis-configuration or via a directory traversal technique. Also, in the rare circumstances the system administrator may have accidentally left a copy of a "passwd" file in a directory accessible for anonymous or other FTP users, which presents a high security risk and simplifies the attack. If the string "passwd" is contained within an otherwise innocuous filename being retrieved from a server,

the rule will generate an event.

Also, the anonymous FTP account often has a separate password file within the chrooted anonymous FTP directory (e.g. /var/ftp/etc/passwd). This file does not usually contain valid system usernames and passwords. While technically not a false positive, this may be considered a false alarm.

Tiny Fragments - Possible Hostile Activity ¹⁹

. This event is generated when an IPv4 fragment of dubious small nature was detected. Many IDS's are known to have issues regarding the reassembly of IP fragments, and could miss an attack carried over such means. Firewalls suffer from the same issues, and can be tricked into allowing packets through that should normally be rejected. Furthermore, there is a small history of OS issues related to unorthodox fragmentation.

IPv4 manages to adapt to various link layer protocols on a route via the fragmentation mechanism outlined in its RFC. A router connecting two carrying media of varying MTU (Maximum Transmission Unit) can fragment packets of size too large to transmit on one wire before dispatch. When datagrams stay within one MTU, the maximum packet sizes possible can be used without fragmentation, thus pairing flexibility with efficiency.

Historically, handling of fragmentation has been less than stellar in both IP stacks and the IDS systems designed to protect them. While the, number of attacks based on fragmentation are easily picked up by anomaly or signature-based system, IDS's which fail to properly reassemble fragments can miss any attack which is so fragmented. Firewalls have often proved susceptible to fragmented TCP or UDP headers, allowing traffic that should have been filtered to pass through. Tools have been written to trivially fragment traffic; Dug Song's fragrouter program is a well-known example.

It is unlikely that such a fragment would be seen in standard use of IPv4; while the last fragment in a series is typically smaller than the others, this signature explicitly matches the More Fragments bit. Nonetheless, a pedantic reading of the IPv4 RFC allows this, so long as the data length is a multiple of 8.

EXPLOIT x86 setgid 0 ²⁰

This event may indicate an exploit attempt where the attacker sent the setgid(0) system call for the x86 platform. This event is specific to a vulnerability, but may have been caused by any of several possible exploits. Signatures used to detect this event are specific and consider the packet payload.

The packet that caused this event is normally a part of an established TCP session, indicating that the source IP address has not been spoofed. If you are

using a firewall that supports state-full inspection, and are not vulnerable to sequence number prediction attacks, then you can be fairly certain that the source IP address of the event is accurate.

There are reported incidents where legitimate traffic may cause an intrusion detection system to raise "false positive" alerts for this event. The following details have been reported: This signature is very short and does not contain context clues to reduce false positives. Therefore, there may be many cases of false alarms where binary data is transferred from outside the network - for example an internal user downloading binary files from an external web server.

DDOS shaft client to handler ²¹

If you must verify that this event represents control traffic, your host may be compromised. Shaft is a distributed denial of service (DDoS) tool. This event is specific to a particular exploit, but the packet payload is not considered as part of the signature to detect the attack.

The packet that caused this event is normally a part of an established TCP session, indicating that the source IP address has not been spoofed. If you are using a firewall that supports statefull inspection, and are not vulnerable to sequence number prediction attacks, then you can be fairly certain that the source IP address of the event is accurate. It has been noted that the intruder is likely to expect or desire a response to their packets, so it may be likely that the source IP address is not spoofed.

There are reported incidents where legitimate traffic may cause an intrusion detection system to raise "false positive" alerts for this event. The following details have been reported: A legitimate server port of 20432 will cause this rule to fire. It may also create a false positive if port 20432 is selected as an FTP data port.

[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.

XDCC

Some servers make it so you do not need to connect to them to view their contents. These are XDCC servers. SPHTML's IRC server has one, a bot (non-human) with the nick "EpisodeMaster". If no one is hosting, or no one has what you want, type "xdcc list" and you should get a list like the one below:

[01:58:57] <plague> xdcc list

...

[01:58:58] -EpisodeMaster- ** To request a file type: "/msg EpisodeMaster xdcc send #x" **

[01:58:58] -EpisodeMaster- #1 19x [37M] South Park Episode 101

[01:58:59] -EpisodeMaster- #2 37x [36M] South Park Episode 612

[01:59:00] -EpisodeMaster- #3 7x [36M] South Park Episode 613

...

To get an episode from an XDCC server, simply type '/msg EpisodeMaster xdcc send #x', as it says, without the quotes. For instance, if you wanted to download Episode 612, you'd type '/msg EpisodeMaster xdcc send #2'. Replace #x with the # of the file.

Probable NMAP fingerprint attempt ²²

Nmap attempts to identify the remote operating system by looking for different services that are common or specific to particular operating systems. It also sends a variety of abnormal packets that are often handled differently by different operating systems so that it can differentiate between them based on the responses. The signature may be produced by other scanners but is unlikely to be used for legitimate activity.

Block any TCP packets that have the SYN, FIN, PUSH and URGENT flags set using a firewall. Block only packets that have all four of the flags set as they are individually and in other combinations necessary for normal TCP traffic. If you block them individually or in other combinations your network will not function correctly.

[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot

XDCC IRC Bot – This bot client is difficult to detect via remote scans because the bot does not use standardized ports in communication with IRC servers. Rather, it comes with a configuration script that allows the hacker to specify (1) which IRC servers to provide service to, (2) what specific channels to join & advertise its availability, (3) the range of ports it should utilize in communication with IRC servers. Furthermore, it uses randomization techniques to bring up a different type of connections of all three described above. A rather poor detection method is via IDS tool such as snort to attempt matching of distinct “signature” string that seems to occur in file sharing transactions between the Bot and the IRC Server. This bot provides file distribution services through the IRC interface, you need to be connected to an IRC Server & associated with a particular channel (or known channel) before you may access the resources that the XDCC bot advertises for availability. Some common access commands are as follows:

To view the list of files available: '/msg [hacked bot] xdcc list'

To request a file package: '/msg [hacked bot] xdcc send #x' (X refers to the package number)

[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot²³

This IRC bot makes use of the IRC facility to distribute files including movies & software. Computers running XDCC may also be remotely controlled by people in the IRC channels. Also, there were cases in which the hacked machines were found to be running a backdoor remote access service, also called XDCC, in addition to the IRC Bot Client described above. This additional backdoor allows the intruder to access the machine as an administrator from a remote computer. Because such access permits ANY commands to be executed on the target machine, the entire data structure, of the victim's machines are at the mercy of the attacker.

Affected Systems - Windows NT/2000, (XP?)

Propagation Mechanism - From the looks of it, currently this exploit is not transmitted via virus, nor worm, but rather a result of directed attack. (however, in no way do we assume that it will stay this way) To our knowledge, the targeted systems were exploited due to the presence of no/weak passwords on the system's administrative accounts.

When it finds a windows machine with file sharing enabled (port 139), it will get a netbios table list of all usernames on that machine. This can be acquired manually also using the following commands in DOS: C:\winnt\system32\>nbtstat -A 127.0.0.1

Where 127.0.0.1 is any given IP with file sharing or netbios. Once it gets a list of all usernames for an IP, it will then check for weak passwords, or no passwords at all. Many people, when installing windows 2000, NT, or XP will forget the true essence of a password. This is highly critical that you set an Administrator password. For people that do not type in a password, this is where it will take advantage of you. It will send back to the attacker the following response:
[127.0.0.1]: Found NT-Server-Password: Administrator/[Blank password]
[127.0.0.1]: "NT-Server-Password" scan complete, Found 1.

Once they have this information, and you have file sharing enabled, consider yourself fully rooted.²⁴

EXPLOIT FTP passwd retrieval retr path²⁵

The attacker downloads a "passwd" file from a machine that does not use shadowed passwords and uses a tool like John-the-Ripper to crack the passwords used for several accounts. He then proceeds to login to the system remote

The attack usually requires FTP access to the /etc/ directory either by system misconfiguration or via a directory traversal technique. Also, in the rare circumstances the system administrator may have accidentally left a copy of a

"passwd" file in a directory accessible for anonymous or other FTP users, which presents a high security risk and simplifies the attack. If the string "passwd" is contained within ly and possibly gain escalated privileges via a local exploit on the system. an otherwise innocuous filename being retrieved from a server, the rule will generate an event.

Back Orifice ²⁶

The operations that the client application can perform on the target machine (e.g., the machine running the server application) include the following:

- Execute any application on the target machine.
- Log keystrokes from the target machine.
- Restart the target machine.
- Lockup the target machine.
- View the contents of any file on the target machine.
- Transfer files to and from the target machine.
- Display the screen saver password of the current user of the target machine.

In order for Back Orifice to work, the server application must be installed on the target machine. This involves executing the server application on the target machine. The server application is a single executable file with a size just over 122 kilobytes.

The specific registry value, which points to the server application is configurable. By doing so, the server application always starts whenever Windows starts, and thus is always active. The application will not appear in the Windows task list. The target machine must be running either Windows 95 or Windows 98. The server application will not run on Windows NT. The target machine must have TCP/IP network capabilities. The client application communicates with the server application using TCP with encrypted UDP packets.

By default, if the server application has not been otherwise configured, the installed filename is ".exe" (e.g., that's a space followed by ".exe"), the communication port is 31337, the registry value name is empty (e.g., the default registry value entry is used), and no password is used (although the communication is still encrypted).

© SANS Institute 2004, Author retains full rights.

logsnorter-0.2

```
#!/usr/bin/perl  
#  
$|=1;
```

```
$DEBUG=0;

use Getopt::Std;
use Sys::Syslog;
use DBI;
use Socket;
use Sys::Hostname;
use Net::hostent;

$VERSION='0.2';

($sec,$min,$hour,$mday,$mon,$year,$yday,$isdst)=localtime(time);
$year += 1900;

$db_server = 'localhost';
$db_database = 'snort';
$db_usercode = 'snort';
$db_password = 'xxx';

$DB_TYPE="mysql";

getopts('thVvu:p:s:d:T:L:');

if ($opt_h) {
    print "logsnorter [-u db_usercode] [-p db_password] [-s db_server]\n";
    print "logsnorter [-d db_database]\n";
    print "logsnorter [-T /var/log/syslog] - to continually 'tail' syslog files\n";
    print "logsnorter [-v] - to show every line (use with \"-T\" and pipe back into swatch)\n";
    print "logsnorter [-L bitmask] - restrict search to known log formats (\"-L -h\" for help)\n";
    print "logsnorter -t <use timestamp of syslog msg instead of current time>\n\n";
    print "System-specific perl code should be put into /etc/logsnorter.conf (e.g. the cisco_interface[] arrays)\n\n";
    exit ;
}

if ($opt_V) {
    print "logsnorter: Version $VERSION\n";
    exit;
}

if ($opt_L && $opt_L !~ /^[0-9]+$/) {
    print "
logformats:
  1   = cisco format
  2   = ipfwadm format
  4   = ipchains format
  8   = iptables format
  16  = SANS log format

```

32 = BSD ipf format

e.g. To get logsnorter to scan for cisco and iptables only, choose

\"-L 9\" #that's 1 + 8

Not choosing the \"-L\" option is equivalent to asking for *all* formats to be scanned for.

```
";  
    exit;  
}
```

```
if ( -f "/etc/logsnorter.conf" ) {  
    require ("/etc/logsnorter.conf" );  
}
```

#Command-line options override the config file...

```
$db_usercode = $opt_u if ($opt_u);  
$db_password = $opt_p if ($opt_p);  
$db_server = $opt_s if ($opt_s);  
$db_database = $opt_d if ($opt_d);
```

```
$ip_ver=4;
```

```
$ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip_csum=0;
```

```
$tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_urp=0;
```

```
$udp_len=$udp_csum=0;
```

```
$icmp_csum=$icmp_id=$icmp_seq=0;
```

```
$hex=0;
```

```
$base64=1;
```

```
$ascii=2;
```

```
$cid=0;
```

```
%month_array = ( 'Jan' => 1,  
                  'Feb' => 2,  
                  'Mar' => 3,  
                  'Apr' => 4,  
                  'May' => 5,  
                  'Jun' => 6,  
                  'Jul' => 7,  
                  'Aug' => 8,  
                  'Sep' => 9,  
                  'Oct' => 10,  
                  'Nov' => 11,  
                  'Dec' => 12);
```

```
$dbh = DBI-
>connect("DBI:$DB_TYPE:database=$db_database;host=$db_server",$db_userc
ode,$db_password) || die "\nCannot access DB server!\n";

if ($opt_T) {
    open(STDIN,"tail -f $opt_T|")||die "cannot tail $opt_T - $!";
}
while (<STDIN) {
    print if ($opt_v);
    $line++;
    if (/ logsnorter: /) {
        next;
    }
    chomp;
    #Get rid of ratty old swatch escape chars if present
    # s/\033\[0m//g;
    # &initialize_vars;

    &parse_cisco_logs if (!$opt_L || $opt_L & 1);
    &parse_ipfwadm_logs if (!$opt_L || $opt_L & 2);
    &parse_ipchains_logs if (!$opt_L || $opt_L & 4);
    &parse_iptables_logs if (!$opt_L || $opt_L & 8);
    #&parse_bastille_logs if (!$opt_L || $opt_L & 16);
    #&parse_ipf_logs if (!$opt_L || $opt_L & 32);
    &parse_sans_logs if (!$opt_L || $opt_L & 16);
}

#added by jrjr
sub parse_sans_logs {
    $logtype='snort';
    $interface_prepend="tensleep_ids_";

    $line=$_;

    # if the line is blank, go to the next one
    next if $line eq "";

    # get Date and time variables if alerts or oos file

    if ( $line =~ m/^(\\d+)\\/(\\d+)\\-(\\d+)\\:(\\d+)\\:(\\d+)\\.(\\d+)\\s/o) {
        $month = $1; $day = $2; $hour = $3; $min = $4; $sec = $5;
        if ($month == 1) {
            $month = 'Jan';
        } elsif ($month == 2) {
            $month = 'Feb';
        } elsif ($month == 3) {
            $month = 'Mar';
        } elsif ($month == 4) {
            $month = 'Apr';
        } elsif ($month == 5) {
            $month = 'May';
        } elsif ($month == 6) {
            $month = 'Jun';
        } elsif ($month == 7) {
            $month = 'Jul';
        } elsif ($month == 8) {
            $month = 'Aug';
        }
    }
}
```

```
} elif ($month == 9) {
    $month = 'Sep';
} elif ($month == 10) {
    $month = 'Oct';
} elif ($month == 11) {
    $month = 'Nov';
} else {
    $month = 'Dec';
}
}

# get Date and time variables if scans file

if ( $line =~ m/^(\\w+)\\s+(\\d+)\\s(\\d+)\\:(\\d+)\\:(\\d+)\\s/o) {
    $month = $1;    $day = $2;    $hour = $3;    $min = $4;    $sec = $5;
}

# is this line an alert message like the following example?
# 06/11-00:00:10.930818  [**] CS WEBSERVER - external web
traffic [**] 65.65.192.18:3745 -> 10.10.100.165:80

if ( $line =~ m/^(\\d+)\\/(\\d+)\\-(\\d+)\\:(\\d+)\\:(\\d+)\\. (\\d+)\\s+
    \\[\\*\\*\\]\\s*(.*)\\s*\\[\\*\\*\\]\\s+
    ([\\d\\.]+)[\\:]*([\\d]*)\\s[\\-\\>]+\\s([\\d\\.]+)[\\:]*([\\d]*)/ox) {
    $ruleset = $7;    $src_addr = $8;
    $src_port = $9; $dst_addr = $10; $dst_port = $11;

    $src_addr =~ /^(\\d+)(\\. (\\d+)(\\. (\\d+)(\\. (\\d+))$)/;
    $array_src_addr[0]=$1;
    $array_src_addr[1]=$2;
    $array_src_addr[2]=$3;
    $array_src_addr[3]=$4;

    $dst_addr =~ /^(\\d+)(\\. (\\d+)(\\. (\\d+)(\\. (\\d+))$)/;
    $array_dst_addr[0]=$1;
    $array_dst_addr[1]=$2;
    $array_dst_addr[2]=$3;
    $array_dst_addr[3]=$4;

    # should check for protocol here (??? udp alerts)
    if ($src_port =~ /\\d+/ && $dst_port =~ /\\d+/) {
        $ip_proto = 6;
    } else {
        #give the type, code and seq a value since there is none in the
data
        $ip_proto = 1;
        $icmp_type = 8;
        $icmp_code = 0;
        $icmp_seq = 0;
    }

    $sensor_ip=10.10.10.10;
    $interface=eth0;
    &find_sensor($sensor_ip,$interface);
    &insert_event;
}
```

```
# are these lines line from an oos file like the following
```

```
#06/26-00:06:04.625435 209.47.197.12:47819 -> MY.NET.25.73:25
#TCP TTL:48 TOS:0x0 ID:4702 IpLen:20 DgmLen:60 DF
#12*****S* Seq: 0xE7C95BFA Ack: 0x0 Win: 0x16D0 TcpLen: 40
#TCP Options (5) => MSS: 1380 SackOK TS: 919440288 0 NOP WS: 0
```

##=+++++
=+=+

```
#06/26-00:06:27.567671 195.14.205.250:21515 ->
```

MY.NET.69.217:3456

```
#12****S* Seq: 0x9A6E3E79 Ack: 0x0 Win: 0x4000 TcpLen: 44
```

3621671 0

```

elseif ( $line =~ m/^(\\d+)\\/(\\d+)\\-(\\d+)\\:(\\d+)\\:(\\d+)\\. (\\d+)\\s+
([\\d\\.]+)[\\:]( [\\d]*)\\s[\\-\\>]+\\s([\\d\\.]+)[\\:]*([\\d]*)/ox)
{

```

```
$src_addr = $7; $src_port = $8; $dst_addr = $9; $dst_port = $10;
```

```

    }
elseif ( $line =~
    ^(\w+)\sTTL:(\d+)\sTOS:0x(\w+)\sID:(\d+)\sIpLen:(\d+)\sDgmLen:(\d+)\s
    (\w+)/ox)
    {
        $ip_sym = $1; $ip_ttl = $2; $ip_tos = $3; $ip_id = $4;
        $ip_hlen = $5; $ip_len = $6; $ip_flags = $7;
    }
}

```

```

    }
    elseif ( $line =~
    ^([1|*][2|*][U|*][A|*][P|*][R|*][S|*][F|*])\sSeq:\s0x(\w+)\s+
    ck:\s0x(\w+)\s+
        Win:\s0x(\w+)\s+TcpLen:\s(\d+)/ox)
    {
        $tcp_flags = $1; $tcp_seq = $2; $tcp_ack = $3; $tcp_win = $4;
        $tcp_len = $5;
        $ruleset = 'out_of_spec';
    }
}

```

```

    #print $month, "/", $day, "-", $hour, ":", $min, ":", $sec, " ";
    #print $src_addr, ":", $src_port, " -> ", $dst_addr, ":",
dst_port, "\n";
    #print $ip_sym, " TTL:", $ip_ttl, " TOS:0x", $ip_tos, " ID:",
p_id, " IpLen:", $ip_hlen,
        "DgmLen:", $ip_len, " ", $ip_flags, "\n";
    #print $tcp_flags, " Seq: 0x", $tcp_seq, " Ack: 0x",
tcp_ack, " Win: 0x", $tcp_win, " TcpLen: ",
        $tcplen, "\n\n";

```

```
# tcp options and data needs to be implemented
```

```
if ($ip_sym =~ /UDP/) {
    $ip_proto = 17;}
elsif ($ip_sym =~ /TCP/) {
    $tcp_off,$tcp_res,$tcp_csum,$tcp_urp=0;
```



```
        $tcp_seq = hex($tcp_seq);
        $tcp_win = hex($tcp_win);
        $tcp_ack = hex($tcp_ack);
        $ip_proto = 6;}
    else { $ip_proto = 1;}

    $flags = 0;
    $flags +=1 if $tcp_flags =~ /F/;
    $flags +=2 if $tcp_flags =~ /S/;
    $flags +=4 if $tcp_flags =~ /R/;
    $flags +=8 if $tcp_flags =~ /P/;
    $flags +=16 if $tcp_flags =~ /A/;
    $flags +=32 if $tcp_flags =~ /U/;
    $flags +=64 if $tcp_flags =~ /2/;
    $flags +=128 if $tcp_flags =~ /1/;
    $tcp_flags=$flags;

    $src_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
    $array_src_addr[0]=$1;
    $array_src_addr[1]=$2;
    $array_src_addr[2]=$3;
    $array_src_addr[3]=$4;

    $dst_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
    $array_dst_addr[0]=$1;
    $array_dst_addr[1]=$2;
    $array_dst_addr[2]=$3;
    $array_dst_addr[3]=$4;

    $sensor_ip=10.10.10.10;
    $interface=eth0;
    &find_sensor($sensor_ip,$interface);
    &insert_event;
}

# or does this line come from a scan file with lines that have
the following format
#Jun 27 00:00:02 my.net.97.169:1029 -> 58.139.93.241:137 UDP
#Jun 27 00:00:02 my.net.97.169:1414 -> 218.30.13.142:80 SYN
*****S*
#Jun 27 00:00:12 217.39.62.204:44391 -> my.net.112.196:80 SYN
12*****S* RESERVEDBITS
#Jun 27 00:00:13 my.net.97.169:1028 -> 192.253.227.47:137 UDP

else { if( $line =~ m/^(\\w+)\\s+(\\d+)\\s(\\d+)\\:(\\d+)\\:(\\d+)\\s+
([\\d\\.]+)[\\:]*([\\d\\.]+)\\s([\\-\\>]+)\\s([\\d\\.]+)[\\:]*([\\d\\.]+)\\s(\\w+)\\s?
([1|\\*][2|\\*][U|\\*][A|\\*][P|\\*][R|\\*][S|\\*][F|\\*])*/ox) {
    $src_addr = $6; $src_port = $7; $dst_addr = $8; $dst_port = $9;
    $ip_proto = $10;
    $ruleset = 'scans'; $tcp_flags = $11;

    $src_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
    $array_src_addr[0]=$1;
    $array_src_addr[1]=$2;
    $array_src_addr[2]=$3;
    $array_src_addr[3]=$4;
```

```
$dst_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)$/;
$array_dst_addr[0]=$1;
$array_dst_addr[1]=$2;
$array_dst_addr[2]=$3;
$array_dst_addr[3]=$4;

if ($ip_proto =~ /UDP/) {
    $ip_proto = 17;
} elsif ($ip_proto =~
/SYN|UNKNOWN|INVALIDACK|NOACK|NULL|VECNA|SYNFIN|XMAS|FULLXMAS|FIN|SPAU|
NMAPID/) {
    $ip_proto = 6;
    $flags = 0;
    $flags +=1 if $tcp_flags =~ /F/;
    $flags +=2 if $tcp_flags =~ /S/;
    $flags +=4 if $tcp_flags =~ /R/;
    $flags +=8 if $tcp_flags =~ /P/;
    $flags +=16 if $tcp_flags =~ /A/;
    $flags +=32 if $tcp_flags =~ /U/;
    $flags +=64 if $tcp_flags =~ /2/;
    $flags +=128 if $tcp_flags =~ /1/;
    $tcp_flags=$flags;
} else { $ip_proto = 1;}

$sensor_ip=10.10.10.10;
$interface=eth0;
&find_sensor($sensor_ip,$interface);
&insert_event;
}
}

sub parse_ipchains_logs {
    my $i,$num_packets;
    &debug("parse_ipchains_logs...");
    $logtype='ipchains';
    $interface_prepend="syslog_";

    $data=$_;
    if (/^(\\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
Packet log:/) {
        /^(\\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
Packet log: ([^\\s]+) (\\w+) (\\w+) PROTO=([0-9]+) ([0-9\\.]+):([0-9]+)
([0-9\\.]+):([0-9]+) L=([0-9]+) S=(\\w+) I=([0-9]+) F=(\\w+) T=([0-9]+)
(.*)$/i;

        $month=$1;
        $day=$2;
        $hour=$3;
        $min=$4;
        $sec=$5;
        $linux=$6;
        $ruleset=$7;
        $auth=$8;
        $interface=$9;
        $ip_proto=$10;
        $src_addr=$11;
```

```
$src_port=$12;
$dst_addr=$13;
$dst_port=$14;
$ip_len=$15;
$ip_tos=$16;
$ip_id=$17;
$frag_off=$18;
$ip_ttl=$19;
$therest=$20;
next if ($auth !~ /REJECT|DENY/);
if ( $linux eq "" ) {
    &crash("Cannot interpret ipchains line (proto=$ip_proto): $_\n");
}
$linux=~s/\s//g;

if ($ip_proto == 6) {
    $therest =~ /\s\(\#\(.*)\)\$/;
    $num_packets=$1;
    $syn="SYN" if ($therest =~ / SYN /);
}
if ( $ip_proto == 1 ) {
    $icmp_type=$src_port;
    $icmp_code=0;
}

$hstip=&getaddr($linux);
if ($hstip ne $sensor_ip) {
    $sensor_ip=$hstip;
    &find_sensor($sensor_ip,$interface);
}

if (!$src_addr || !$dst_addr || !$ip_proto ) {
    &crash("IPChains Error line $line: insufficient info - broken
syslog entry!");
}

$src_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_src_addr[0]=$1;
$array_src_addr[1]=$2;
$array_src_addr[2]=$3;
$array_src_addr[3]=$4;

$dst_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_dst_addr[0]=$1;
$array_dst_addr[1]=$2;
$array_dst_addr[2]=$3;
$array_dst_addr[3]=$4;

#IPChains doesn't count packets - i.e. one syslog == 1 packet
$num_packets=1 if (!$num_packets);
for ($i=0;$i<$num_packets;$i++) {
    &debug("i=$i,num_packets=$num_packets");
    &insert_event;
    next;
}
```

```
}  
}  
  
sub parse_ipfwadm_logs {  
    &debug("parse_ipfwadm_logs");  
    $logtype='ipfwadm';  
    $interface_prepend="syslog_";  
    $num_packets=1;  
  
    $data=$_;  
    if (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^s]+) kernel: IP  
fw/) {  
        $ip_ver=4;  
  
        $ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip  
_csum=0;  
  
        $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_  
urp=0;  
        $udp_len=$udp_csum=0;  
        $icmp_csum=$icmp_id=$icmp_seq=0;  
  
        if (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^s]+) kernel:  
IP fw-(\w+) (\w+) ([:\w]+) ([\w\/]+) ([0-9\.] +):([0-9]+) ([0-  
9\.] +):([0-9]+) L=([0-9]+) S=(\w+) I=([0-9]+) F=(\w+) T=([0-9]+)/i) {  
            $month=$1;  
            $day=$2;  
            $hour=$3;  
            $min=$4;  
            $sec=$5;  
            $linux=$6;  
            $ruleset=$7;  
            $auth=$8;  
            $interface=$9;  
            $proto=$10;  
            $src_addr=$11;  
            $src_port=$12;  
            $dst_addr=$13;  
            $dst_port=$14;  
            $ip_len=$15;  
            $ip_tos=$16;  
            $ip_id=$17;  
            $frag_off=$18;  
            $ip_ttl=$19;  
            $therest=$20;  
            next if ($auth ne "rej");  
        } elseif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^s]+)  
kernel: IP fw-(\w+) (\w+) ([:\w]+) ([\w\/]+) ([0-9\.] +) ([0-9\.] +)  
L=([0-9]+) S=(\w+) I=([0-9]+) F=(\w+) T=([0-9]+)/i) {  
            $month=$1;  
            $day=$2;  
            $hour=$3;  
            $min=$4;  
            $sec=$5;  
            $linux=$6;  
            $ruleset=$7;
```

```
$auth=$8;
$interface=$9;
$proto=$10;
$src_addr=$11;
$dst_addr=$12;
$ip_len=$13;
$ip_tos=$14;
$ip_id=$15;
$frag_off=$16;
$ip_ttl=$17;
$therest=$18;
next if ($auth ne "rej");
} else {
    &crash("Unknown ipfw match on line $line: $_");
}
if ($proto =~ /^(.*)\/(.*$/) {
    $proto=$1;
    $type=$2;
}
&get_ip_proto($proto);
if ( $linux eq "" ) {
    &crash("Cannot interpret ipfw line (proto=$ip_proto): $_\n");
} else {
}
$linux=~s/\'/\'/g;

if ($ip_proto == 6) {
    $therest =~ /\s\(\#(.*)\)$/;
}
if ( $ip_proto == 1 ) {
    $icmp_type=$type;
    $icmp_code=0;
}

$hstip=&getaddr($linux);
if ($hstip ne $sensor_ip) {
    $sensor_ip=$hstip;
    &find_sensor($sensor_ip,$interface);
}

if (!$src_addr || !$dst_addr || !$ip_proto ) {
    &crash("IPFWadm Error line $line: insufficient info - broken syslog
entry!");
}

$src_addr=~/^([0-9]+\)\.([0-9]+\)\.([0-9]+\)\.([0-9]+\)$/;
$array_src_addr[0]=$1;
$array_src_addr[1]=$2;
$array_src_addr[2]=$3;
$array_src_addr[3]=$4;

$dst_addr=~/^([0-9]+\)\.([0-9]+\)\.([0-9]+\)\.([0-9]+\)$/;
$array_dst_addr[0]=$1;
$array_dst_addr[1]=$2;
$array_dst_addr[2]=$3;
$array_dst_addr[3]=$4;
```

```
    for ($i=0;$i<=$num_packets;$i++) {
        &insert_event;
        next;
    }
}

sub parse_cisco_logs {
    &debug("parse_cisco_logs");
    $logtype='ciscoacl';
    $interface_prepend="syslog_";

    $data=$_;

    if (/IPACCESSLOG/i) {
        $ip_ver=4;

        $ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip_
        _csum=0;

        $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_
        urp=0;
        $udp_len=$udp_csum=0;
        $icmp_csum=$icmp_id=$icmp_seq=0;

        if (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) ([0-
        9]+): ([^\\s]+) .*IPACCESSLOG: list .* (udp|tcp)/) {
            /^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) ([0-9]+):
            ([^\\s]+) ([^\\s]+): list ([0-9]+) (\\w+) (\\w+) ([0-9\\.]+)\\(((0-9)+)\\)
            (.*)([0-9\\.]+)\\(((0-9)+)\\), ([0-9]+) packet/i;

            $month=$1;
            $day=$2;
            $hour=$3;
            $min=$4;
            $sec=$5;
            $cisco=$6;
            $junk=$7;
            $junk1=$8;
            $tag=$9;
            $ruleset=$10;
            $auth=$11;
            $proto=$12;
            $src_addr=$13;
            $src_port=$14;
            $interface=$15;
            $dst_addr=$16;
            $dst_port=$17;
            $num_packets=$18;
            next if ($auth ne "denied");
            if ( $cisco eq "" ) {
                &crash("Cannot interpret Cisco line (proto=$proto): $_\\n");
            }
            $cisco=~s/\\/\\/g;
            $hstip=&getaddr($cisco);
            if ($interface =~ /^(\\s+)([\\s\\s]+)\\) ([^\\s]+$/)) {
```

```
$interface = $1;
} else {
  if (!${cisco_interface[${cisco},${ruleset}]) {
    #&crash("Cannot interpret Cisco line as no interface
found.\nDefine array \${cisco_interface[${cisco},${ruleset}] =
'FastEthernet0/0' or whatever in $0, then re-run.\n$_");
    ${cisco_interface[${cisco},${ruleset}]="logsnorter";
  }
  $interface=${cisco_interface[${cisco},${ruleset}];
}
if ($hstip ne $sensor_ip) {
  $sensor_ip=$hstip;
  &find_sensor($sensor_ip,$interface);
}

} elsif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+)
([0-9]+): ([^\s]+) .*IPACCESSLOGDP: list .* icmp/) {
  /^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+) ([0-9]+):
([^\s]+) ([^\s]+): list ([0-9]+) (\w+) (\w+) ([0-9\.]+) \((([^\s]+)\)\)
[^\s]+ ([0-9\.]+) \((([0-9]+)\)/([0-9]+)\), ([0-9]+) packet/i;

  $month=$1;
  $day=$2;
  $hour=$3;
  $min=$4;
  $sec=$5;
  $cisco=$6;
  $cisco=~s/\'/\'/g;
  $junk=$7;
  $junk1=$8;
  $tag=$9;
  $ruleset=$10;
  $auth=$11;
  $proto=$12;
  $src_addr=$13;
  $interface=$14;
  $dst_addr=$15;
  $icmp_type=$16;
  $icmp_code=$17;
  $num_packets=$18;
  next if ($auth ne "denied");
  if ( ${cisco} eq "" ) {
    &crash("Cannot interpret icmp Cisco line (proto=$proto): $_\n");
  }
  $cisco=~s/\'/\'/g;
  $hstip=&getaddr($cisco);
  if ($hstip ne $sensor_ip) {
    $sensor_ip=$hstip;
    &find_sensor($sensor_ip,$interface);
  }
} elsif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+)
([0-9]+): ([^\s]+) ([^\s]+)IPACCESSLOGS: list ([0-9]+) (\w+) ([0-9\.]+)
([0-9]+) packet/i) {

  /^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+) ([0-9]+):
([^\s]+) ([^\s]+): list ([0-9]+) (\w+) ([0-9\.]+) ([0-9]+) packet/i;
  $month=$1;
```

```
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$cisco=$6;
$cisco=~s/\'/\'/g;
$junk=$7;
$junk1=$8;
$tag=$9;
$ruleset=$10;
$auth=$11;
$src_addr=$12;
$num_packets=$13;
next if ($auth ne "denied");
if ( $cisco eq "" ) {
&crash("Cannot interpret icmp Cisco line (proto=$proto): $_\n");
}
$cisco=~s/\'/\'/g;
$hstip=&getaddr($cisco);
$dst_addr=$hstip;
$ip_proto="0";
if ($hstip ne $sensor_ip) {
$sensor_ip=$hstip;
&find_sensor($sensor_ip,$interface);
}
} else {
&crash("Cisco error line $line: doesn't match known type: $_\n");
}

#Convert proto back to number
&get_ip_proto($proto);

if (!$src_addr || !$dst_addr || $ip_proto eq "" || ($ip_proto == 1
&& $icmp_type eq "")) || !$ruleset) {
&crash("Cisco Error line $line: insufficient info - broken syslog
entry!
(src_addr=$src_addr,dst_addr=$dst_addr,ip_proto=$ip_proto,icmp_type=$ic
mp_type,acl=$ruleset)");
}

$src_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_src_addr[0]=$1;
$array_src_addr[1]=$2;
$array_src_addr[2]=$3;
$array_src_addr[3]=$4;

$dst_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_dst_addr[0]=$1;
$array_dst_addr[1]=$2;
$array_dst_addr[2]=$3;
$array_dst_addr[3]=$4;

if ($num_packets == 0) {
&crash("Cisco Whaa! No number of packets for line $line -
duh...");
}
```



```
&insert_event;
next;
}
}

#function added by siafu
sub parse_iptables_logs {
    &debug("parse_iptables_logs");
    $logtype='iptables';
    $interface_prepend="syslog_";

    $data=$_;
    if (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
IN=/) {
        $ip_ver=4;

        $ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip
_csum=0;

        $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_
urp=0;
        $udp_len=$udp_csum=0;
        $icmp_csum=$icmp_id=$icmp_seq=0;
        if (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
IN=(\\w+[0-9]+) OUT= SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+)
TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+) PROTO=(\\w+) SPT=([0-9]+)
DPT=([0-9]+) WINDOW=([0-9]+) RES=(\\w+) ([\\w+\\s]+)URGP=([1-9]+)/) {
            #If we audited an incoming TCP packet
            $month=$1;
            $day=$2;
            $hour=$3;
            $min=$4;
            $sec=$5;
            $linux=$6;
            $ruleset=IN;
            $auth=audit;
            $interface=$7;
            $src_addr=$8;
            $dst_addr=$9;
            $ip_len=$10;
            $ip_tos=$11;
            # PREC=$12
            $ip_ttl=$13;
            $ip_id=$14;
            $proto=$15;
            $src_port=$16;
            $dst_port=$17;
            # WINDOW=$18;
            # RES=$19;
            # FLAGS=$20
            # URGP=$21
            $therest=$22;
            # $frag_off=$;
        } elseif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: IN= OUT=(\\w+[0-9]+) SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+)
TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+) PROTO=(\\w+) SPT=([0-9]+)
DPT=([0-9]+) WINDOW=([0-9]+) RES=(\\w+) ([\\w+\\s]+)URGP=([0-9]+)/) {
```

```
#If we audited an outgoing TCP packet
$month=$1;
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$linux=$6;
$ruleset=OUT;
$auth=audit;
$interface=$7;
$src_addr=$8;
$dst_addr=$9;
$ip_len=$10;
$ip_tos=$11;
# PREC=$12
$ip_ttl=$13;
$ip_id=$14;
$proto=$15;
$src_port=$16;
$dst_port=$17;
# WINDOW=$18;
# RES=$19;
# FLAGS=$20
# URGP=$21
$therest=$22;
# $frag_off=$;
} elseif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: IN=(\\w+[0-9]+) OUT= SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+)
TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+) DF PROTO=(\\w+) SPT=([0-
9]+) DPT=([0-9]+) LEN=([0-9]+)/) {
    #If we audited an incoming UDP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=OUT;
    $auth=audit;
    $interface=$7;
    $src_addr=$8;
    $dst_addr=$9;
    $ip_len=$10;
    $ip_tos=$11;
    # PREC=$12
    $ip_ttl=$13;
    $ip_id=$14;
    $proto=$15;
    $src_port=$16;
    $dst_port=$17;
    # LEN=$18;
    $therest=$18;
    # $frag_off=$;
    } elseif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: IN= OUT=(\\w+[0-9]+) SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+)
TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+) DF PROTO=(\\w+) SPT=([0-
9]+) DPT=([0-9]+) LEN=([0-9]+)/) {
```

```
#If we audited an outgoing UDP packet
$month=$1;
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$linux=$6;
$ruleset=OUT;
$auth=audit;
$interface=$7;
$src_addr=$8;
$dst_addr=$9;
$ip_len=$10;
$ip_tos=$11;
# PREC=$12
$ip_ttl=$13;
$ip_id=$14;
$proto=$15;
$src_port=$16;
$dst_port=$17;
# LEN=$18;
$therest=$18;
# $frag_off=$;
#Jun 20 14:24:13 fw kernel: IN=eth0 OUT=eth1 SRC=xxx.xxx.xxx.xxx
DST=xxx.xxx.xxx.xxx LEN=40 TOS=0x00 PREC=0x00 TTL=127 ID=186 DF
PROTO=TCP SPT=3021 DPT=80 WINDOW=5840 RES=0x00 ACK URGP=0
} elseif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+)
kernel: IN=(\w+[0-9]+) OUT=(\w+[0-9]+) SRC=([0-9\.]+) DST=([0-9\.]+)
LEN=([0-9]+) TOS=(\w+) PREC=(\w+) TTL=([0-9]+) ID=([0-9]+) DF
PROTO=(\w+) SPT=([0-9]+) DPT=([0-9]+) WINDOW=([0-9]+) RES=(\w+)
([\w+\s]+)URGP=([0-9]+)) {
    #If we audited a FORWARD TCP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=FORWARD;
    $auth=audit;
    #Interface-IN=$7
    #Interface-OUT=$8
    $interface=$8;
    $src_addr=$9;
    $dst_addr=$10;
    $ip_len=$11;
    $ip_tos=$12;
    # PREC=$13
    $ip_ttl=$14;
    $ip_id=$15;
    $proto=$16;
    $src_port=$17;
    $dst_port=$18;
    # WINDOW=$19;
    # RES=$20;
    # FLAGS=$21
    # URGP=$22
```

```
$therest=$23;
# $frag_off=$;
} elif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+)
kernel: IN=(\w+[0-9]+) OUT=(\w+[0-9]+) SRC=([0-9\.]+) DST=([0-9\.]+)
LEN=([0-9]+) TOS=(\w+) PREC=(\w+) TTL=([0-9]+) ID=([0-9]+) DF
PROTO=(\w+) SPT=([0-9]+) DPT=([0-9]+) LEN=([0-9]+)/) {
    #If we audited a FORWARD UDP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=FORWARD;
    $auth=audit;
    #Interface-IN=$7
    #Interface-OUT=$8
    $interface=$8;
    $src_addr=$9;
    $dst_addr=$10;
    $ip_len=$11;
    $ip_tos=$12;
    # PREC=$13
    $ip_ttl=$14;
    $ip_id=$15;
    $proto=$16;
    $src_port=$17;
    $dst_port=$18;
    # LEN=$19;
    $therest=$19
} else {
    &crash("Unknown iptables match on line $line: $_");
}
&parse_iptables_vars;
}

sub parse_iptables_vars {
    &debug("parse_iptables_vars");
    if ($proto =~ /^(.*)\/(.*)$/) {
        $proto=$1;
        $type=$2;
    }
    &get_ip_proto($proto);
    if ( $linux eq "" ) {
        &crash("Cannot interpret iptables line (proto=$ip_proto): $_\n");
    }
    $linux=~s/\'/\'/g;

    if ($ip_proto == 6) {
        $therest =~ /\s\(\#(.*)\)\$/;
    }
    if ( $ip_proto == 1 ) {
        $icmp_type=$type;
        $icmp_code=0;
    }
}
```

```
$hstip=&getaddr($linux);
if ($hstip ne $sensor_ip) {
    $sensor_ip=$hstip;
    &find_sensor($sensor_ip,$interface);
}

if (!$src_addr || !$dst_addr || !$ip_proto ) {
    &crash("IPTables Error line $line: insufficient info - broken syslog
entry!");
}

$src_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_src_addr[0]=$1;
$array_src_addr[1]=$2;
$array_src_addr[2]=$3;
$array_src_addr[3]=$4;

$dst_addr=~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/;
$array_dst_addr[0]=$1;
$array_dst_addr[1]=$2;
$array_dst_addr[2]=$3;
$array_dst_addr[3]=$4;

#IPTables doesn't count packets - i.e. one syslog == 1 packet
&insert_event;
next;
}

#function added by siafu
#Bastille Firewall prepends 'audit' to the syslog message when logging
specific ports
sub parse_bastille_auditlogs {
    &debug("parse_bastille_auditlogs");
    $logtype='bastille';
    $interface_prepend="syslog_";

    $data=$_;
    # Bastille regular audit logs
    if (/^(\\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
audit/) {
        $ip_ver=4;

        $ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip
_csum=0;

        $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp
urp=0;
        $udp_len=$udp_csum=0;
        $icmp_csum=$icmp_id=$icmp_seq=0;

        if (/^(\\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
auditIN=(\\w+[0-9]+) OUT= MAC=([a-f,0-9,:]+) SRC=([0-9\\.]+) DST=([0-
9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+) DF
PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+) WINDOW=([0-9]+) RES=(\\w+) SYN
URG=([0-9]+)) {
            #If we audited an incoming TCP packet
            $month=$1;
```

```
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$linux=$6;
$ruleset=IN;
$auth=audit;
$interface=$7;
$mac=$8;
$src_addr=$9;
$dst_addr=$10;
$ip_len=$11;
$ip_tos=$12;
# PREC=$13
$ip_ttl=$14;
$ip_id=$15;
$proto=$16;
$src_port=$17;
$dst_port=$18;
# WINDOW=$19;
# RES=$20;
# SYN URGP=$21
$therest=$22;
# $frag_off=$;
} elif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: auditIN= OUT=(\\w+[0-9]+) MAC=([a-f,0-9,:]+) SRC=([0-9\\.]+)
DST=([0-9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-
9]+) DF PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+) WINDOW=([0-9]+) RES=(\\w+)
SYN URGP=([0-9]+)/) {
    #If we audited an outgoing TCP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=OUT;
    $auth=audit;
    $interface=$7;
    $mac=$8;
    $src_addr=$9;
    $dst_addr=$10;
    $ip_len=$11;
    $ip_tos=$12;
    # PREC=$13
    $ip_ttl=$14;
    $ip_id=$15;
    $proto=$16;
    $src_port=$17;
    $dst_port=$18;
    # WINDOW=$19;
    # RES=$20;
    # SYN URGP=$21
    $therest=$22;
    # $frag_off=$;
} elif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: auditIN=(\\w+[0-9]+) OUT= MAC=([a-f,0-9,:]+) SRC=([0-9\\.]+)
```

```
DST=([0-9\.]+) LEN=([0-9]+) TOS=(\w+) PREC=(\w+) TTL=([0-9]+) ID=([0-9]+)
PROTO=(\w+) SPT=([0-9]+) DPT=([0-9]+) LEN=([0-9]+)/) {
    #If we audited an incoming UDP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=IN;
    $auth=audit;
    $interface=$7;
    $mac=$8;
    $src_addr=$9;
    $dst_addr=$10;
    $ip_len=$11;
    $ip_tos=$12;
    # PREC=$13
    $ip_ttl=$14;
    $ip_id=$15;
    $proto=$16;
    $src_port=$17;
    $dst_port=$18;
    # LEN=$19;
    $therest=$20;
    # $frag_off=$;
} elseif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: auditIN= OUT=(\\w+[0-9]+) MAC=([a-f,0-9,:]+) SRC=([0-9\\.]+)
DST=([0-9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+) ID=([0-9]+)
PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+) LEN=([0-9]+)/) {
    #If we audited an outgoing UDP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=OUT;
    $auth=audit;
    $interface=$7;
    $mac=$8;
    $src_addr=$9;
    $dst_addr=$10;
    $ip_len=$11;
    $ip_tos=$12;
    # PREC=$13
    $ip_ttl=$14;
    $ip_id=$15;
    $proto=$16;
    $src_port=$17;
    $dst_port=$18;
    # LEN=$19;
    $therest=$20;
    # $frag_off=$;
} else {
    &crash("Unknown iptables match on line $line: $_");
}
```

```
&parse_iptables_vars;
}
}

# function added by siafu
# Bastille Firewall prepends 'PUB_IN','PUB_OUT','INT_IN', 'INPUT' to
syslogs when
# logging all packets that are denied or rejected.
sub parse_bastille_denylogs {
    &debug("parse_bastille_denylogs");
    $logtype='bastille';
    $interface_prepend="syslog_";

    $data=$_;
    if ((/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
PUB_(\\w+)/) || (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: INT_(\\w+)/) || (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+)
([^\\s]+) kernel: INPUT (\\w+)/)) {
        $ip_ver=4;

        $ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_proto=$ip
_csum=0;

        $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_
urp=0;
        $udp_len=$udp_csum=0;
        $icmp_csum=$icmp_id=$icmp_seq=0;
        if (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+) kernel:
(\\w+) (\\w+) ([0-9]+) IN=(\\w+[0-9]+) OUT= MAC=(\\w+[a-f,0-9,:]+) SRC=([0-
9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+) TTL=([0-9]+)
ID=([0-9]+) DF PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+) WINDOW=([0-9]+)
RES=(\\w+) SYN URGP=([0-9]+)/) {
            # If we denied (and logged) an incoming TCP packet
            $month=$1;
            $day=$2;
            $hour=$3;
            $min=$4;
            $sec=$5;
            $linux=$6;
            $ruleset=$7;
            $auth=$8;
            # unknown=$9
            $interface=$10;
            $mac=$11;
            $src_addr=$12;
            $dst_addr=$13;
            $ip_len=$14;
            $ip_tos=$15;
            # PREC=$16
            $ip_ttl=$17;
            $ip_id=$18;
            $proto=$19;
            $src_port=$20;
            $dst_port=$21;
            # WINDOW=$22;
            # RES=$23;
            # SYN URGP=$24
```



```
$therest=$25;
# $frag_off=$;
} elif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: (\\w+) (\\w+) ([0-9]+) IN= OUT=(\\w+[0-9]+) MAC=([a-f,0-9,:]+)
SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+)
TTL=([0-9]+) ID=([0-9]+) DF PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+)
WINDOW=([0-9]+) RES=(\\w+) SYN URGP=([0-9]+)/) {
# If we denied (and logged) an outgoing TCP packet
$month=$1;
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$linux=$6;
$ruleset=$7;
$auth=$8;
# unknown=$9
$interface=$10;
$mac=$11;
$src_addr=$12;
$dst_addr=$13;
$ip_len=$14;
$ip_tos=$15;
# PREC=$16
$ip_ttl=$17;
$ip_id=$18;
$proto=$19;
$src_port=$20;
$dst_port=$21;
# WINDOW=$22;
# RES=$23;
# SYN URGP=$24
$therest=$25;
# $frag_off=$;
} elif (/^(\\w+)\\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\\s]+)
kernel: (\\w+) (\\w+) ([0-9]+) IN=(\\w+[0-9]+) OUT= MAC=([a-f,0-9,:]+)
SRC=([0-9\\.]+) DST=([0-9\\.]+) LEN=([0-9]+) TOS=(\\w+) PREC=(\\w+)
TTL=([0-9]+) ID=([0-9]+) PROTO=(\\w+) SPT=([0-9]+) DPT=([0-9]+) LEN=([0-
9]+)/) {
# If we denied (and logged) an incoming UDP packet
$month=$1;
$day=$2;
$hour=$3;
$min=$4;
$sec=$5;
$linux=$6;
$ruleset=$7;
$auth=$8;
# unknown=$9
$interface=$10;
$mac=$11;
$src_addr=$12;
$dst_addr=$13;
$ip_len=$14;
$ip_tos=$15;
# PREC=$16
$ip_ttl=$17;
```

```
$ip_id=$18;
$proto=$19;
$src_port=$20;
$dst_port=$21;
# LEN=$22;
$therest=$23;
} elsif (/^(\w+)\s+([0-9]+) ([0-9]+):([0-9]+):([0-9]+) ([^\s]+)
kernel: (\w+) (\w+) ([0-9]+) IN= OUT=(\w+[0-9]+) MAC=([a-f,0-9,:]+)
SRC=([0-9\.]+) DST=([0-9\.]+) LEN=([0-9]+) TOS=(\w+) PREC=(\w+)
TTL=([0-9]+) ID=([0-9]+) PROTO=(\w+) SPT=([0-9]+) DPT=([0-9]+) LEN=([0-
9]+))/) {
    # If we denied (and logged) an outgoing UDP packet
    $month=$1;
    $day=$2;
    $hour=$3;
    $min=$4;
    $sec=$5;
    $linux=$6;
    $ruleset=$7;
    $auth=$8;
    # unknown=$9
    $interface=$10;
    $mac=$11;
    $src_addr=$12;
    $dst_addr=$13;
    $ip_len=$14;
    $ip_tos=$15;
    # PREC=$16
    $ip_ttl=$17;
    $ip_id=$18;
    $proto=$19;
    $src_port=$20;
    $dst_port=$21;
    # LEN=$22;
    $therest=$23;
} else {
    &crash("Unknown iptables match on line $line: $_");
}
&parse_iptables_vars;
}
}

sub find_sensor {
    my ($hst,$int) = @_;

    #Glean the interface from $interface
    if ($int =~ /\s/) {
        $int =~ /^(([^\s]+)\s)/;
        $int = $1;
    }
    if (!$int) {
        &crash("Error line $line: no interface found\n");
    }
}
```

```
$sth = $dbh->prepare(q{select sid from sensor where hostname = ? and  
interface = ?});  
$src = $sth->execute($hst,"$interface_prepend$int") || &crash("DB  
Error on line $line: \"select sid from sensor where hostname = '$hst'\"  
failed - $!");  
($sid) = $sth->fetchrow_array;  
if (!$sid) {  
    #It's a new one!  
    $sth = $dbh->prepare(q{InSeRt INTO sensor (hostname, interface,  
detail, encoding) VALUES (?,?,'1',?)});  
    $src = $sth->execute($hst,"$interface_prepend$int",$ascii) ||  
&crash("DB Error on line $line: \"insert into sensor hostname=$hst\"  
failed - $!");  
  
    $sth = $dbh->prepare(q{select sid from sensor where hostname = ?  
and interface = ?});  
    $src = $sth->execute($hst,"$interface_prepend$int") || &crash("DB  
Error on line $line: \"select sid from sensor where hostname = '$hst'\"  
failed - $!");  
    ($sid) = $sth->fetchrow_array;  
    $cid = 0;  
} else {  
    #It's an existing sensor - find current max cid  
    &find_max_cid($sid);  
}  
}  
  
sub find_signature {  
    my ($sig)=@_  
    $sth = $dbh->prepare(q{SELECT sig_id FROM signature WHERE sig_name =  
?});  
    $src = $sth->execute($sig)|| &crash("DB Error on line $line: \"SELECT  
sig_id FROM signature WHERE sig_name = '$sig'\" failed - $!");  
    ($sig_num) = $sth->fetchrow_array;  
    # print "find_sig[$rec_count]: sig_num=$sig_num,sig_name=$sig\n";  
    if (!$sig_num) {  
    # print "find_sig[$rec_count]: insert into signature (sig_name)  
Values $sig\n";  
        $sth = $dbh->prepare(q{InSeRt INTO signature (sig_name) VALUES  
(?)});  
        $src = $sth->execute($sig)|| &crash("DB Error on line $line:  
\"InSeRt INTO signature (sig_name) VALUES ('$sig')\" failed - $!");  
        #Recurse - but check count!  
        $rec_count++;  
        if ($rec_count > 2) {die "Infinite loop forming in find_signature  
call! Dying..."};  
        &find_signature($sig);  
    }  
    $rec_count=0;  
    return($sig_num);  
}  
  
sub get_ip_proto {  
    my ($proto)=@_  
    if ($proto =~ /^[0-9]+$/) {  
        $ip_proto=$proto  
    } else {
```

```
my $ip_proto_name,$ip_proto_alias,$ip_proto_num;

($ip_proto_name,$ip_proto_alias,$ip_proto_num)=getprotobyname($proto);
$ip_proto=$ip_proto_num;
}
}

sub find_max_cid {
    my ($id) = @_;
    $sth = $dbh->prepare(q{select max(cid) from event where sid = ?});
    $src = $sth->execute($id) || &crash("DB Error on line $line: \"select
max(cid) from event where sid = '$id'\" failed - $!");
    ($cid) = $sth->fetchrow_array;
}

sub insert_event {
    my $i=0;
    &debug("insert_event called...");
    $cid++;
    my $logevent, $src_type,$dst_type,$src_port_name, $dst_port_name ;

    &crash("Broken record - sid not defined!") if (!$sid);
    &crash ("Bogus interface definition on line $line") if ($interface =~
/\-\>|\<\-|\<\/);

    if ($ip_proto != 1) {
        $src_port_name=&getservice($src_port,$ip_proto);
        if ($src_port > 1023) {
            if ($src_port eq $src_port_name) {
                $src_type='high';
            } else {
                $src_type=$src_port_name;
            }
        } else {
            $src_type=$src_port_name;
        }
        $dst_port_name=&getservice($dst_port,$ip_proto);
        if ($dst_port > 1023) {
            if ($dst_port_name eq $dst_port) {
                $dst_type="high";
            } else {
                $dst_type=$dst_port_name;
            }
        } else {
            $dst_type = $dst_port_name;
        }
    } else {
        $src_type=$dst_type='icmp';
    }
    # $logevent="$logtype-$ruleset/$src_type->$dst_type"; (too long)
    if ((length $ruleset) > 255) {
        $tmp=truncate $ruleset, 255;
        $logevent=$tmp;
    } else { $logevent="$ruleset"; }

    # next if ( $logevent =~ /$ignore_stupid_matches/);
```

```
$signature=&find_signature($logevent);
if ($opt_t) {
    $sth = $dbh->prepare(q{InSeRt INTO event
(sid,cid,signature,timestamp) VALUES (?, ?, ?, ?)});
    $src = $sth->execute($sid,$cid,$signature, "$year-
".$month_array{$month}."-$day $hour:$min:$sec") || &crash("DB Error on
line $line: \"InSeRt INTO event (sid,cid,signature,timestamp) VALUES
($sid,$cid,'$signature', '$year-".$month_array{$month}."-$day
$hour:$min:$sec')\" failed - $!");
} else {
    $sth = $dbh->prepare(q{InSeRt INTO event
(sid,cid,signature,timestamp) VALUES (?, ?, ?, NOW())});
    $src = $sth->execute($sid,$cid,"$signature") || &crash("DB Error on
line $line: \"InSeRt INTO event (sid,cid,signature,timestamp) VALUES
($sid,$cid,'$signature',NOW())\" failed - $!");
}
#Generate int32 versions of ip addresses...
$src_int32=$dst_int32='0x';
for ($z=0;$z<4;$z++) {
    $src_int32 .= sprintf "%2x",$array_src_addr[$z];
    $dst_int32 .= sprintf "%2x",$array_dst_addr[$z];
}
$src_int32=~ s/\s/0/g;
$dst_int32=~ s/\s/0/g;
$ip_src= hex($src_int32);
$ip_dst= hex($dst_int32);
$sth = $dbh->prepare(q{InSeRt InTO iphdr (sid, cid, ip_src,
ip_dst,ip_ver,ip_hlen, ip_tos, ip_len, ip_id, ip_flags, ip_off,ip_ttl,
ip_proto, ip_csum) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)});
$src = $sth->execute($sid,$cid,$ip_src, $ip_dst, $ip_ver,$ip_hlen,
$ip_tos, $ip_len, $ip_id, $ip_flags, $ip_off, $ip_ttl, $ip_proto,
$ip_csum) || &crash("DB Error on line $line: \"InSeRt INTO iphdr with
sid=$sid, cid=$cid\" failed - $!");
if ( $ip_proto == 6) {
    $sth = $dbh->prepare(q{InSeRt INTO tcphdr (sid, cid, tcp_sport,
tcp_dport, tcp_seq, tcp_ack, tcp_off, tcp_res, tcp_flags, tcp_win,
tcp_csum, tcp_urp) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)});
    $src = $sth->execute($sid,$cid,$src_port, $dst_port,$tcp_seq,
$tcp_ack, $tcp_off, $tcp_res, $tcp_flags, $tcp_win, $tcp_csum,
$tcp_urp) || &crash("DB Error on line $line: \"InSeRt INTO tcphdr with
sid=$sid, cid=$cid\" failed - $!");
}
if ( $ip_proto == 17) {
    $sth = $dbh->prepare(q{InSeRt INTO udphdr (sid, cid, udp_sport,
udp_dport, udp_len, udp_csum) VALUES (?, ?, ?, ?, ?, ?)});
    $src = $sth->execute($sid,$cid,$src_port, $dst_port,$udp_len,
$udp_csum) || &crash("DB Error on line $line: \"InSeRt INTO udphdr with
sid=$sid,cid=$cid\" failed - $!");
}
if ( $ip_proto == 1) {
    $sth = $dbh->prepare(q{InSeRt INTO icmphdr (sid, cid, icmp_type,
icmp_code, icmp_csum, icmp_id, icmp_seq) VALUES (?, ?, ?, ?, ?, ?, ?)});
    $src = $sth->execute($sid,$cid,$icmp_type,$icmp_code,$icmp_csum,
$icmp_id, $icmp_seq) || &crash("DB Error on line $line: \"InSeRt INTO
icmphdr with sid=$sid,cid=$cid\" failed - $!");
}
$data =~ s/\n/ /g;
```

```
$data =~ s/\'/\'/g;
$sth = $dbh->prepare(q{InSeRt INTO data (sid,cid,data_payload) VALUES
(?,?,?)});
$src = $sth->execute($sid,$cid,$data) || &crash("DB Error on line
$line: \"InSeRt INTO data with sid=$sid,cid=$cid,data=$data\" failed -
$!");
}

sub crash {
    my ($entry) = @_;
    $entry="$entry\n" if ( $entry !~ /\n$/ );
    syslog('info', "logsnorter: $entry");
    print "logsnorter: Error line $line. $entry";
    if (!$opt_T ) {
        exit;
    } else {
        next;
    }
}

sub initialize_vars {
    $ip_ver=4;
    $src_port=$dst_port='';

    $proto=$ip_hlen=$ip_tos=$ip_len=$ip_id=$ip_flags=$ip_off=$ip_ttl=$ip_pr
oto=$ip_csum=0;

    $tcp_seq=$tcp_ack=$tcp_off=$tcp_res=$tcp_flags=$tcp_win=$tcp_csum=$tcp_
urp=0;
    $udp_len=$udp_csum=$i=$z=0;
    $icmp_csum=$icmp_id=$icmp_seq=0;
}

sub getaddr {
    my ($hn)=@_;
    my $h,$ip;
    $h = gethostbyname($hn)||&crash("getaddr: Cannot resolve $hn to IP
address! Cannot run logsnorter on a host with different DNS or
/etc/hosts entries than the host that generated this syslog message!");
    $ip=inet_ntoa($h->addr);
    return($ip);
}

sub getservice {
    my ($a,$b)=@_;
    my $h;
    my($proto,$alias,$num);
    ($proto,$alias,$num)=getprotobyname($b);
    $h = getservbyport($a,$proto);
    if ($h) {
        return($h);
    } else {
        return($a);
    }
}
```

```
sub debug {  
    if ($DEBUG) {  
        print @_, "\n";  
    }  
}
```

__END__

=head1 NAME

I<logsnorter> - scans syslog messages for HIDS messages and pumps them into a SQL snort database

=head2 SYNOPSIS

```
logsnorter [-vVt] -T /var/log/syslog -u user \  
    -p passwd -s dbserver -d database
```

=head1 DESCRIPTION

This perl script scans syslog messages looking for reports of packets denied by router/firewall/HIDS systems (such as Cisco routers), translates them into snort format and injects them into the same SQL database system B<snort> is running on.

Currently the following systems are supported:

=over 2

=item o Cisco access-lists

=item o Linux 2.0 ipfw

=item o Linux 2.2 ipchains

=item o Linux 2.4 iptables

=back

=back

Merging information like perimeter router data into the central B<snort>

NIDS system allows you to gain a more comprehensive report of what attacks and scans are being done against your network. If your perimeter router is blocking all packets besides port 25 and port 80, B<snort> (sitting behind that router) will not even know that there are full-blown port scans being run against your network. Using I<logsnorter> to merge those messages back into the B<snort> database allows such events to be seen.

The wonderful PHP-based B<snort> analysis package - I<ACID> - is invaluable

in this regard. See [B<http://www.cert.org/kb/acid/>](http://www.cert.org/kb/acid/) for details.

=head1 OPTIONS

=over 2

=item -u DB username

=item -p DB password

=item -d DB database

=item -s DB server

=item -v

Verbose: print every syslog line to STDOUT as it is scanned for matches. This option can be used to "chain" logsnorter in front of swatch (i.e. swatch calls "logsnorter -T /var/log/syslog" instead of "tail -f /var/log/syslog").

=item -t

Use the timestamps associated with the syslog message instead of the current time. This must be used when post-processing syslog files (e.g. running logsnorter once per day over yesterdays syslog messages) otherwise all the entries going into the snort database will have the current time! Also be sure all your systems are using NTP or the like so that their timestamps are in sync.

=item -T

"tail" the filename that follows - typically /var/log/syslog. This is the most used form for logsnorter, where it is permanently left running, continually scanning incoming syslog messages and pumping them back into the snort database in real-time.

=back

=back

=head1 CONFIGURATION

As there are quite a few command-line options, these are better placed in the config file /etc/logsnorter.conf. This file is simply merged back into the actual logsnorter script. i.e. it must contain valid perl commands:

=head2 Example Configuration File

#This is /etc/logsnorter.conf

\$db_server='localhost';

\$db_usercode='snort';


```
$db_database='snort';

$db_password='9f54b53j954';

#Cisco access-list syslog messages don't report the interface
#which generated the message. You must therefore provide logsnorter
#with this information (indexed to the ACL number) so that it can
#correctly inject these into the snort database

$cisco_interface['rtr01',107]="Serial0.1";
$cisco_interface['rtr01',108]="Serial0.1";
$cisco_interface['rtr11',105]="FastEthernet0";
$cisco_interface['rtr11',106]="FastEthernet0";

=head1 TODO

=over 2

=item o Support for PostgreSQL? It may already work - someone tell me!

=item o Support for non-Linux packet filters - e.g. BSD ipfilter.
Someone else will have to do it. Code welcome!

=back
=back

=head1 AUTHOR

Jason Haar <jhaar@users.sourceforge.net>

=cut
```

process_scanalert.pl

```
#!/usr/local/bin/perl -ws
#
# use: process_scanalert.pl -v -l=1 samplescan.data > analysis.output
#
unless
(defined($c)||defined($d)||defined($o)||defined($p)||defined($s)||defin
ed($t)||defined($u)||defined($v)){
    print "use the following action flags\n";
    print "\t-c \tprint the communicating hosts\n";
    print "\t-d \tprint the target hosts\n";
    print "\t-f \twatch for fingerprinting attempts\n";
```

```
        print "\t-o \tprint the attacked/scanned ports\n";
        print "\t-p \tprint the attacker/target pair\n";
        print "\t-s \tprint the attacking hosts\n";
        print "\t-t \tprint the attack type\n";
        print "\t-u \tprint summary from spp_portscan\n";
        print "\t-v \tbe verbose and print everything\n";
        print "----";
        print "\t-l=n\tconnection threshold before printing\n";
        exit 1;
    }
}
if(defined($l)){ $thresh=$l;}else{$thresh=0;}
$vv=0; #set $vv=1 for very verbose output for script debugging

#####
# Data parsing and analysis #
#####
while (< >){
    #format of scan.YYMMDD
    if($ _ =~ /^( [a-zA-Z]{3} \s \d{1,2} ) \s ( [0-9\:]{8} ) \s ( [0-9\.] + ) \s ( ([0-9\w]+) \s \- \> \s ( [0-9\.] + ) \s ( ([0-9\w]+) \s ( \w+ [ \w\s]* )*) ) /){
        chomp;
        $date=$1;$time=$2;$src_addr=$3;$src_port=$4;
        $dst_addr=$5;$dst_port=$6;$scantype=$7;
        $pkey = "$src_addr-$dst_addr";
        $pkeyl = "$src_addr:$src_port-$dst_addr:$dst_port";
        ++$asrc{$src_addr}; #count the same source address
        ++$adst{$dst_addr}; #count the same dest address
        if($scantype =~ /UDP/){
            ++$pdst_udp{$dst_port};} #count the same dest port
        else{++$pdst_tcp{$dst_port};}
        ++$type{$scantype}; #count the same scan type
        ++$pair{$pkey}; #count the same connection direction (IP only)
        ++$pairl{$pkeyl}; #count the same connection direction (IP:port)
        #look for OS fingerprint scans
        unless (( $scantype =~ /SYN\s\*\*\*\*\*\*S\*/ ) || ( $scantype =~ /UDP/ ) || ( $scantype =~ /FIN\s\*\*\*\*\*\*F\* ) ){
            ++$fsrc{$src_addr};
            ++$fdst{$dst_addr};
            if($scantype != /UDP/){++$fpdst_tcp{$dst_port};}
            ++$fpr{$pkey};
            ++$ftyp{$scantype};}
        }
    #format of alert.YYMMDD
    elsif($ _ =~ /^( [0-9\ /]{5} \- [0-9\:\.]{15} ) \s+ \[ \*\*\* \] /){
        #look for portscans
        if($ _ =~ /portscan/i){
            #we investigate only the summary of the portscans
            if($ _ =~ /^( [0-9\ /]{5} ) \- ( [0-9\:\.]{15} ) \s+ \[ \*\*\* \] \s+ spp_portscan \: \s+ End \sof \sportscan \s+ from \s ( ([0-9\.] + ) \s+ \ ( TOTAL \sHOSTS \: ( ([0-9\*]+ ) \s+ TCP \: ( ([0-9\*]+ ) \s+ UDP \: ( ([0-9\*]+ ) \) \) \s+ \[ \*\*\* \] /){
                $date=$1;$time=$2;$portscan_dst=$3;
                $nmb_portscanners=$4;$nmb_scanTCPpacks=$5;$nmb_scanUDPPacks=$6;
                ++$type{"Spp_portscan detect"};
                $nmb_portscan{$portscan_dst} += $nmb_portscanners;
                if($nmb_scanTCPpacks != 0){
                    $portscan_tcp{$portscan_dst} += $nmb_scanTCPpacks;}
```

```

        if($nmb_scanUDPPacks != 0){
            $portscan_udp{$portscan_dst}+=$nmb_scanUDPPacks;}
        }
        else{if($vv){print"Ignore intermediate output from spp_portscan
$_ \n";}}}
    }
    #parse 'normal' alerts
    else{ #need (.*) stingy instead of greedy matching here...
        if($_ =~ /^(([0-9\/]{5})\)-([0-9\:\.]{15})\s+\[\\*\]\s+(.*)\s+\[\\*\]\s+([0-9\.]+\)\:([0-9w]+\)\s\-\>\s+([0-9\.]+\)\)\:([0-9w]+\)\)/){
            $date=$1;$time=$2;$attacktype=$3;$src_addr=$4;$src_port=$5;
            $dst_addr=$6;$dst_port=$7;
            $pkey = "$src_addr-$dst_addr";
            $pkeyl = "$src_addr:$src_port-$dst_addr:$dst_port";
            ++$asrc{$src_addr};
            ++$adst{$dst_addr};
            ++$pdst{$dst_port}; #cannot distinguish udp/tcp here easily
            if((! defined($portlist{$attacktype}))||
                ($portlist{$attacktype} !~ /$dst_port/)){
                $portlist{$attacktype} .= " $dst_port";}
            ++$type{$attacktype};
            ++$pair{$pkey};          #count the same connection direction (IP
only)
            ++$pairl{$pkeyl};}      #count the same connection direction
(IP:port)
        elsif($_ =~ /^(([0-9\/]{5})\)-([0-9\:\.]{15})\s+\[\\*\]\s+(.*)\s+\[\\*\]\s+([0-9\.]+\)\s+\-\>\s+([0-9\.]+\)\)/){
            #signatures that don't involve ports, e.g. ICMP traffic
            $date=$1;$time=$2;$attacktype=$3;$src_addr=$4;$dst_addr=$5;
            $pkey = "$src_addr-$dst_addr";
            ++$asrc{$src_addr};
            ++$adst{$dst_addr};
            ++$type{$attacktype};
            ++$pair{$pkey};
        }
        else{
            if($v){
                print"Ignore line $_ \n";}}}
    }
}
else{print "!!! Ignore unknown data: $_ \n";}
}

#####
# Print various statistics #
#####
#Print different scan signatures
if (($t)||($v)&&(defined(%type))){
    print "\nAttack/Scan Types\n===== \n\n" if ($v) ;
    foreach $key (sort { $type{$b} <=> $type{$a} }keys(%type)){
        if($v){ if ($key =~ /\s*(.*)\s+/){ $totaltype{$1}+= $type{$key};}}
        if(((($f)||($v))&&defined($ftyp{$key})){ $fp="\t\t($ftyp{$key}
fps\);}
        else{ $fp=""; }
        print "$type{$key} \t\t$key $fp\n" if($type{$key} >= $thresh);

```

```

    }
    print "\n\Total Scan Types\n===== \n\n" if ($v) ;
    foreach $key (sort { $totaltype{$b} <=> $totaltype{$a} } keys(%totaltype)){
        print "$totaltype{$key} \t$key \n" if($totaltype{$key} >= $thresh);
    }
}
#Print target hosts
if (($d)||($v)&&(defined(%adst))){
    print "\n\nTargets\n===== \n\n" if ($v) ;
    foreach $key (sort { $adst{$b} <=> $adst{$a} } keys(%adst)){
        if((($f)||($v))&&($fdst{$key})){$fp="\t($fdst{$key}
fps)";}else{$fp="";}
        if(($key =~ /\.255\s*$/)||($key =~ /\.0\s*$/)){ $BCAST=" BCAST
";}else{ $BCAST="";}
        if($key
=~ /\^s*(224|225|226|227|228|229|230|231|232|233|234|235|236|237|238|239
)\.\/){ $MCAST=" MCAST ";}else{ $MCAST="";}
        print "$adst{$key} \t$key $fp $MCAST $BCAST\n" if($adst{$key} >=
$thresh);
    }
}
if (($f)||($v)&&(defined(%fdst))){
    print "\n\nFingerprinted targets\n===== \n\n" if ($v)
;
    foreach $key (sort { $fdst{$b} <=> $fdst{$a} } keys(%fdst)){
        $fp="fps";
        if(($key =~ /\.255\s*$/)||($key =~ /\.0\s*$/)){ $BCAST=" BCAST
";}else{ $BCAST="";}
        if($key
=~ /\^s*(224|225|226|227|228|229|230|231|232|233|234|235|236|237|238|239
)\.\/){ $MCAST=" MCAST ";}else{ $MCAST="";}
        print "$fdst{$key} \t$key $fp $MCAST $BCAST\n" if($adst{$key} >=
$thresh);
    }
}
#Print attacking hosts
if (($s)||($v)&&(defined(%asrc))){
    print "\n\nAttackers by frequency\n===== \n\n" if
($v) ;
    foreach $key (sort { $asrc{$b} <=> $asrc{$a} } keys(%asrc)){
        if((($f)||($v))&&($fsrc{$key})){$fp="\t($fsrc{$key}
fps)";}else{$fp="";}
        if(($key =~ /\^s*10\.\/)||($key
=~ /\^s*172\.(16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31)\.\/)||($ke
y =~ /\^s*192\.(168\.\.\/)){ $SPOOFED=" SPOOF ";}else{ $SPOOFED="";}
        print "$asrc{$key} \t $key $fp $SPOOFED\n" if($asrc{$key} >=
$thresh);}
        if($vv){print "\n\nAttackers by IP\n===== \n\n" if
($v) ;
            foreach $key (sort { $a cmp $b } keys(%asrc)){
                if((($f)||($v))&&($fsrc{$key})){$fp="\t($fsrc{$key}
fps)";}else{$fp="";}
                print "$asrc{$key} \t$key $fp\n" if($asrc{$key} >= $thresh);}}
    }
    if (($f)||($v)&&(defined(%fdst))){

```

```
print "\n\nFingerprinting attackers\n===== \n\n" if ($v) ;
foreach $key (sort { $fsrc{$b} <=> $fsrc{$a} } keys(%fsrc)){
    $fp="fps";
    if(($key =~/^s*10\.\/)||($key =~/^s*172\.(16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31)\.\/)||($key =~/^s*192\.168\.\/)){ $SPOOFED=" SPOOF ";}else{ $SPOOFED="";}
    print "$fsrc{$key} \t $key $fp $SPOOFED\n" if($fsrc{$key} >= $thresh);
}
}
#Print summary from spp_portscan
if (($u)||($v)&&(defined(%nmb_portscan))){
    print "\n\nspp_portscan statistics\n===== \n\n" if ($v) ;
    print "* Nmb. of portscans\n";
    foreach $key (sort { $nmb_portscan{$b} <=> $nmb_portscan{$a} } keys(%nmb_portscan)){
        print"$nmb_portscan{$key} \t$key \n" if($nmb_portscan{$key} >= $thresh);}
    if(defined(%portscan_tcp)){
        print "* Nmb. of tcp packets\n";
        foreach $key (sort { $portscan_tcp{$b} <=> $portscan_tcp{$a} } keys(%portscan_tcp)){
            print"$portscan_tcp{$key} \t$key \n" if($portscan_tcp{$key} >= $thresh);}
        }
    if(defined(%portscan_udp)){
        print "* Nmb. of udp packets\n";
        foreach $key (sort { $portscan_udp{$b} <=> $portscan_udp{$a} } keys(%portscan_udp)){
            print"$portscan_udp{$key} \t$key \n" if($portscan_udp{$key} >= $thresh);}
        }
    }
}
#Print attacked/scanned ports
if (($o)||($v)&&(defined(%pdst_tcp))){
    print "\n\nScanned TCP ports\n===== \n\n" if ($v) ;
    foreach $key (sort { $pdst_tcp{$b} <=> $pdst_tcp{$a} } keys(%pdst_tcp)){
        if((($f)||($v))&&($fpdst_tcp{$key})){ $fp="\t($fpdst_tcp{$key} fps)";}else{ $fp="";}
        print "$pdst_tcp{$key} \t$key $fp\n" if($pdst_tcp{$key} >= $thresh);
    }
}
if (($o)||($v)&&(defined(%pdst_udp))){
    print "\n\nScanned UDP ports\n===== \n\n" if ($v) ;
    foreach $key (sort { $pdst_udp{$b} <=> $pdst_udp{$a} } keys(%pdst_udp)){
        print "$pdst_udp{$key} \t$key \n" if($pdst_udp{$key} >= $thresh);}
    }
}
if (($o)||($v)&&(defined(%pdst))){
    print "\n\nPorts in alert files\n===== \n\n" if ($v) ;
    foreach $key (sort { $pdst{$b} <=> $pdst{$a} } keys(%pdst)){
        print "$pdst{$key} \t$key \n" if($pdst{$key} >= $thresh);}
    }
}
```

```
if (($o)||($v)&&(defined(%portlist))){
    print "\n\nPorts by attack\n===== \n\n" if ($v) ;
    foreach $key (sort keys(%portlist)){
        print "$key \t $portlist{$key} \n";
    }
}
#Attacker-Target pairs
if (($p)||($v)&&(defined(%pair))){
    print "\n\nAttacks/Targets\n===== \n\n" if ($v) ;
    foreach $key (sort { $pair{$b} <=> $pair{$a} } keys(%pair)){
        if((($f)||($v))&&($fpr{$key})){ $fp="\t($fpr{$key}
fps)"; }else{ $fp=""; }
        print "$pair{$key} \t $key $fp\n" if($pair{$key}>= $thresh);
    }
}
#Communicating parties
if (($c)||($v)&&(defined(%pair))){
    print "\n\nCommunicating parties\n===== \n\n" if ($v)
;
    foreach $key (sort { $pair{$b} <=> $pair{$a} } keys(%pair)){
        if($key =~ /([0-9\.\+])\-([0-9\.\+])/){
            $src=$1;$dst=$2;
            $revkey="$dst-$src";
            if(defined($pair{$revkey})){
                if($src le $dst) { $commkey="$src-$dst"; }
                else { $commkey="$dst-$src"; }
                if((($f)||($v))&&($fpr{$key})){ $fp="\t(fp)"; }else{ $fp=""; }
                if($key eq $commkey)
                    { $commpair{$commkey} .= " -> $pair{$key} $fp \*\*\* "; }
                else
                    { $commpair{$commkey} .= " <- $pair{$key} $fp \*\*\* "; }
            }
        }
    }
    foreach $commkey (sort keys(%commpair)){
        print "$commkey \*\*\* $commpair{$commkey} \n";
    }
}
```

¹ C0ldPhaTe. "Microsoft IIS Unicode Exploit Explained." URL:
<http://www.astalavista.com/library/os/iis/>

- ² Tiffany Bergeron. "Post to OVAL-DISCUSSION-LIST. Open Vulnerability Assessment Language." Jan 2003. URL: <http://oval.mitre.org/community/forum/archives/2003-01/msg00010.html>
- ³ Brian Caswell. Judy Novak. Snort Signature Database. "Documentation for rule WEB-IIS unicode directory traversal attempt." URL: <http://www.snort.org/snort-db/sid.html?sid=1945>
- ⁴ Jim Forster. Subject: Re: [snort] 'SMB Name Wildcard' Date: Mon Jan 17 2000 - 09:26:14 CST URL: <http://archives.neohapsis.com/archives/snort/2000-01/0222.html>
- ⁵ TonikGin. "XDCC – An .EDU Admin's Nightmare. Sept. 11 2002." URL: <http://www.russonline.net/tonikgin/eduhacking.html>
- ⁶ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center." "Advisory CA-2001-23 Continued Threat of the "Code Red" Worm." URL: <http://www.cert.org/advisories/CA-2001-23.html>
- ⁷ Gunther Birznies. "Web Application Security." URL: http://www.extropia.com/presentations/birznies/pdf/cgi_security_history.pdf
- ⁸ Joe Stewart. In reply to: Len Burns: "[Snort-users] CGI Null Byte Attack" Nov. 2000. URL: <http://archives.neohapsis.com/archives/snort/2000-11/0244.html>
- ⁹ Jon Hart. Snort Signature Database. "Documentation for rule SHELLCODE x86 NOOP." URL: <http://www.snort.org/snort-db/sid.html?sid=648>
- ¹⁰ Carnegie Mellon Software Engineering Institute. "CERT Coordination Center." "CERT® Advisory CA-2003-10 Integer overflow in Sun RPC XDR library routines." URL: <http://www.cert.org/advisories/CA-2003-10.html>
- ¹¹ Max Vision. Brian Caswell. Judy Novak. Snort Signature Database. "Documentation for rule RPC portmap listing TCP 111." URL: <http://www.snort.org/snort-db/sid.html?sid=598>
- ¹² Microsoft Corporation. Microsoft Security Bulletin MS01-033L-098 "Microsoft Index Server ISAPI Extension Buffer Overflow". June 2001. URL: <http://www.ciac.org/ciac/bulletins/l-098.shtml>
- ¹³ LinuxSecurity.com Features. "Scanning and Defending Networks with Nmap."

URL: http://www.linuxsecurity.com/feature_stories/feature_story-4.html

¹⁴ Nigel Houghton. Josh Sakofsky. Snort Signature Database. “Documentation for rule NETBIOS SMB C\$ access.” URL: <http://www.snort.org/snort-db/sid.html?sid=533>

¹⁵ Carnegie Mellon Software Engineering Institute. “CERT Coordination Center.” “CERT® Incident Note IN-2000-02 Exploitation of Unprotected Windows Networking Shares.” URL: http://www.cert.org/incident_notes/IN-2000-02.html

¹⁶ Jon Hart. Snort Signature Database. “Documentation for rule SHELLCODE x86 setuid 0.” URL: <http://www.snort.org/snort-db/sid.html?sid=650>

¹⁷ Matt Kettler. Snort Signature Database. “Documentation for rule SHELLCODE x86 stealth NOOP.” URL: <http://www.snort.org/snort-db/sid.html?sid=651>

¹⁸ Max Vision. Anton Chuvakin. Nigel Houghton. Snort Signature Database. “Documentation for rule FTP passwd retrieval attempt.” URL: <http://www.snort.org/snort-db/sid.html?sid=356>

¹⁹ Nigel Houghton. Nick Black. Snort Signature Database. “Documentation for rule MISC Tiny Fragments.” URL: <http://www.snort.org/snort-db/sid.html?sid=522>

²⁰ Arachnids - The Intrusion Event Database. “Documentation for rule IDS284/SHELLCODE_SHELLCODE-X86-SETGID0.” URL: <http://www.digitaltrust.it/arachnids/IDS284/event.html>

²¹ Arachnids The Intrusion Event Database. “Documentation for rule IDS254/DDOS_DDOS-SHAFT-CLIENT-TO-HANDLER.” URL: <http://www.digitaltrust.it/arachnids/IDS254/event.html>

²² Nigel Houghton. Steven Alexander. Snort Signature Database. “Documentation for rule SCAN nmap fingerprint attempt.” URL: <http://www.snort.org/snort-db/sid.html?sid=629>

²³ Duke University. “OIT Security.” May 2002
URL: <http://www.oit.duke.edu/security/cleaning/xdcc.html>

²⁴ TonikGin. XDCC – An .EDU Admin’s Nightmare. Sept. 11 2002.
URL: <http://www.russonline.net/tonikgin/eduhacking.html>

²⁵ Max Vision. Anton Chuvakin. Nigel Houghton. Snort Signature Database. “Documentation for rule FTP passwd retrieval attempt.”
URL: <http://www.snort.org/snort-db/sid.html?sid=356>

²⁶ Symantec Security Updates. Information on Back Orifice and NetBus
URL: <http://www.symantec.com/avcenter/warn/backorifice.html>

© SANS Institute 2004, Author retains full rights.