GIAC
CERTIFICATIONS

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 permited 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

GCIA Certified Intrusion Analyst (GCIA) Practical Assignment

Paper version 1.0

Assignment version 3.4 (revised September 24, 2003)
Part 1 option 1

Location of Course Work:
SANS Darling Harbour, Sydney Australia, January 2004

Prepared by: Ian Eaton

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

IPv6 in the Wild

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1 Introduction

This paper provides a summary of the IPv6[1] protocol and an example of its use by black hats in the real world. The intention is not to provide just a technical treatment of IPv6 but provide enough information to understand an IPv6 network trace or at a minimum be able to locate the information required. A real network trace freely available on the Internet is used to introduce IPv6 to the reader, it also helps to limit the scope of the paper to those aspects directly related to the network trace and introduce ways to investigate the source of the attack using similar methods used for IPv4 network traces.

This is a lofty goal, especially when we have limited space and this is my first look into the world of IPv6. A lot of information beyond what I consider important for our discussion will be omitted or only briefly discussed, I hope leaving enough references behind to provide the reader the ability to fill in the gaps I may have left.

1.1 Target Audience

This paper can only be a brief introduction to IPv6 due to length restrictions so the target audience is a technical one. If you have very little experience with IPv4 then you may find it difficult to follow, I expect the reader to have a good grounding in IPv4 with the intention that if you are familiar with IPv4 you should get a lot out of the paper.

1.2 Why have this paper

As IPv6 moves from the testing stage to deployment[2] it is worth taking a looking into the protocol as it becomes more widespread, the IDS analyst needs to become familiar with this protocol as should become apparent when we examine the real world trace.

A complete switch to IPv6 will not happen over night, the short to medium term will see both IPv4 and IPv6 working alongside each other. Today we find pools of IPv6 networks around the world interconnected by a greater ocean of IPv4. These IPv6 pools talk to each other by tunnelling their packets within an IPv4 packet. This way you can connect any computer that has an IPv6 stack to any other computer or IPv6 network. It is actually very easy to tunnel IPv6 leaving great potential for this activity to evade detection, almost acting as a covert channel that only a well-informed IDS analyst will notice.

2 The source of the trace

The real world example used in this paper is available on the internet from the Honeynet Project who provides resources for the security community to learn and teach others about Internet security issues. This trace shows the compromise of a honeypot that later involved the use of IPv6.

2.1 What is a Honeypot and the Honeynet project

A honeypot as described by Lance Spitzner[3] is:

```
An information system resource whose value lies in unauthorized or
illicit use of that resource.
```

A honeypot can take many forms, their existence, or any services they provide are not generally advertised publicly. Attackers locate systems using many techniques, one way is to use a scattergun approach and fire packets randomly across the Internet in the hope

[1] The core IPv6 RFC is RFC2460
[2] RFC3701 discusses the phasing out of the 6bone network which was a test bed for IPv6 activity; this RFC makes mention that allocation of production address prefixes has been underway since 1999.
[3] A more detailed explanation of a honeypot/honeynet – http://www.spitzner.net/honeypots.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

of locating an active host. A host is poked and prodded until a weakness is found or considered too hard for the attacker to compromise.

When there is no public information about a system, any traffic it receives is automatically suspicious, this traffic is recorded in full and later analysed by experts to learn the tricks and techniques of the black hat community. Honeynets take the next step and provide multiple Honeypots that form a networked environment hosted on the Internet.

2.2 Scan of the Month Challenges

The "Scan of the Month Challenge" (SOTM)[4] is one of the services the Honeynet Project provides for anyone to enter. These are challenges that range in required skill level from beginner to advance with the aim of improving the skill base of security practitioners around the world. Simply, most of the challenges involve a piece of information, network trace, Trojan binary file, or a whole system image with a series of questions that an entrant is required to answer. With the top 30 responses posted on the Internet for others to learn from the top few can receive a prize.

2.2.1 Scan of the Month Challenge 28

On the main SOTM page, challenge number 28[5] looked very interesting as the challenger is presented with these words:

```
Scan 28 - Italian blackhats break into a Solaris server then enable
IPv6 tunnelling for communications.
```

This challenge showed there is a real threat with IPv6 and highlights the need for people to learn more about this protocol[6], this attack became the catalyst for improvements to security tools such as snort[7] as the threat had now been realised. This trace is downloadable from the scan of the month28 main page consisting of two files, `day1.log` and `day3.log`. Our main interest is with `day3.log`

One of the ways to identify the Solaris machine is by the syslog traffic, looking at the `day3.log` file in Ethereal at lines `117206`, `117211`, and `117334` we can determine the OS as "`SunOS Release 5.8 Version Generic_108528-09 64-bit`", the system type as "`Sun Ultra 5/10 UPA/PCI (UltraSPARC-IIi 360MHz)`" and the host name "`zoberius`".

3 The IPv6 protocol

The IPv6 protocol is not new, as a technology it has been evolving for many years[8]:

```
IPv6 was recommended by the IPng Area Directors of the Internet
Engineering Task Force at the Toronto IETF meeting on July 25, 1994
in RFC 1752, The Recommendation for the IP Next Generation Protocol.
The recommendation was approved by the Internet Engineering Steering
Group and made a Proposed Standard on November 17, 1994.

The core set of IPv6 protocols were made an IETF Draft Standard on
August 10, 1998.
```

The IPv6 protocol is now beginning to see serious deployment throughout Asian[9] like Japan, Korea and China. In addition, the US defence Department have stipulated they will convert their network infrastructure to IPv6 by 2008[10]. Even more reason to be on top of

---

[4] Scan of the Month Challenge home page – http://www.honeynet.org/scans/index.html.
[5] SOTM Challenge 28 main page – http://www.honeynet.org/scans/scan28/
[6] Newsgroup posting by Lance Spitzner regarding IPv6 and the compromise – http://www.securityfocus.com/archive/119/303782/2002-12-15/2002-12-21/0
[7] Response to Lance Spitzner's posting by Marty Roach, the creator of Snort – http://www.securityfocus.com/archive/119/304374/2002-12-22/2002-12-28/0
[8] This quote was taken from the introduction at http://playground.sun.com/pub/ipng/html/ipng-main.html
[9] IPv6 rollout on horizon for the 'China Next Generation Internet' – http://www.pcpro.co.uk/?http://www.pcpro.co.uk/news/news_story.php?id=55116
[10] Defense Department Will Require IPv6 Compliance – http://biz.yahoo.com/iw/030626/054991.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

this impending technology change. This section introduces IPv6, showing the differences from IPv4, issues with migrating and a detailed look at the protocols construction.

3.1 Major differences from IPv4

IPv4 has been around a long time providing real world experience, developers have a good understanding on what works, and what features are commonly required. RFC2460 titled "Internet Protocol, Version 6 (IPv6) Specification"[11] describes the major changes between IPv4 and IPv6 best:

1. *Expanded Addressing Capabilities;*
   IPv6 increases the IP address size from 32 bits to 128 bits, to support more levels of addressing hierarchy, a much greater number of addressable nodes, and simpler auto-configuration of addresses. The scalability of multicast routing is improved by adding a "scope" field to multicast addresses. In addition, a new type of address called an "anycast address" is defined, used to send a packet to any one of a group of nodes.
2. *Header Format Simplification;*
   Some IPv4 header fields have been dropped or made optional, to reduce the common-case processing cost of packet handling and to limit the bandwidth cost of the IPv6 header.
3. *Improved Support for Extensions and Options;*
   Changes in the way IP header options are encoded allows for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.
4. *Flow Labelling Capability;*
   A new capability is added to enable the labelling of packets belonging to particular traffic "flows" for which the sender requests special handling, such as non-default quality of service or "real-time" service.

   The idea behind this capability[12] is to reduce overhead on Internet routers for packets that require special handling like real-time flows. After passing a packet containing a non-zero flow label any further packets with a similar flow label will be handled in the same way. The assumption is that all packets with the same flow label have the same IPv6 details, i.e. Destination Address, Hop-by-Hop Options header, Routing Header and Source Address contents, continuos inspection of these fields in not necessary.[13]
5. *Authentication and Privacy Capabilities;*
   Extensions to support authentication, data integrity, and (optional) data confidentiality are specified for IPv6.

Some other points of interest are:

6. *Changes to packet fragmentation;*
   Due to improved MTU discovery[14] capabilities, fragmentation is not expected in IPv6. There are no fragmentation fields in the main header, they have been placed into extension headers for the rare occasion they are needed. Fragmentation is only performed by the source, if intermediate hosts (i.e. routers) need to forward a packet along a path with a smaller MTU than was initially intended, it will send an ICMPv6[15] "Packet Too Big" error back to the source who will have to either create smaller packets or fragment the packet itself. Fragmented packets are expected to be reassembled by the destination only.

[11] Full text is in RFC2460
[12] The philosophy behind the Flow label is described in RFC1809 "Using the Flow Label Field in IPv6"
[13] The final specification is in RFC3697
[14] RFC1981 "Path MTU discovery for IP version 6"
[15] ICMPv6 is described in RFC2463

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

7. *Calculation of Checksums in upper level protocols;*
   TCP and UDP use an IPv4 pseudo header when calculating their checksums, these upper layer protocols require a change to incorporate the IPv6 addresses in its calculation.
8. *UDP is now **required** to compute a checksum on all packets;*
9. *ICMPv6 **requires** the calculation of a checksum using the same pseudo header; and*
10. *IPv6 **requires** every link to have an MTU of 1280 bytes or greater.*
    Any node not implementing Path MTU discovery should use 1280 bytes as their maximum packet size, this ensures that fragmentation will not become an issue for the host. If a node generates packets that are smaller than the Path MTU, they could be considered as wasting network resources.

3.2 Compatibility with IPv6

Most major network and OS vendors provide IPv6 support in their products, a list of which vendors and their level of support can be found:
http://playground.sun.com/pub/ipng/html/ipng-implementations.html

While many applications provide IPv6 support to varying degrees, this is an excellent link on application support for Linux and *BSD based services and applications:
http://www.deepspace6.net/docs/ipv6_status_page_apps.html

A lot of security tools also provide IPv6 compatibility, tools used in the analysis of the trace in this paper are Ethereal[16] and tcpdump[17]. IDS products with IPv6 support include snort[18] and ISS[19], each having a varying degree support.

Our targeted Honeypot was identified as a Solaris 8 system, Solaris 8 and above have native IPv6 compatibility that is installed either during initial OS setup or manually if required later. Manual configuration is detailed in Volume 3 of the "System Administration Guide for Solaris 8"[20] on page 351 these are the abridged steps:

1. become superuser
2. touch `/etc/hostname6.`*`interface`*
   where interface is each network interface you would like to have IPv6 enabled
3. Reboot the system
4. Configure name servers as required.

The reboot is necessary because[21]:

```
<it> sends out router discovery packets and the router responds with
a prefix, enabling the node to configure the interfaces with an IP
address. Rebooting also restarts key networking daemons in IPv6 mode.
```

The router will only provide a prefix if configured to do so, though there are other ways to obtain an IP address, including router discovery packets, some address types are configured automatically by the host, and the administrator can manually configure them. One of the most important issues a reboot ensures is all IPv6 capable network daemons are restarted in IPv6 mode, and can start offering services via IPv6 addresses.

[16] Ethereal home page http://www.ethereal.com
[17] Tcpdump homepage http://tcpdump.org/
[18] Snort home page http://www.snort.org
[19] ISS home page http://www.iss.net
[20] To locate this book go to the sun Production Documentation site at http://docs.sun.com/db/prod/solaris and search for "system administration guide volume 3", and click on "Search book titles only".
[21] Volume 3 of the "System Administration Guide for Solaris 8" page 251

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.3 IPv6 migration path

IPv6 will affect every host and piece of infrastructure residing directly on the Internet. Knowing this the IETF in parallel with designing the protocol itself they have been working on technologies and processes to make the transition as smooth as possible, which is going to be long as the Internet is inherently IPv4 based.

As pockets of IPv6 networks or individual IPv6 capable hosts spring up randomly across the Internet, there needs to be a means for seamless communication between them even when separated by IPv4 networks. Tunnelling IPv6 packets inside IPv4 is an important transition mechanism, this allows communication to travel across the Internet while requiring no modification to the underlying IPv4 network infrastructure.

There are many IPv6 transition methods available today with others in progress moving through the standards body[22] [23]. This list only shows some of the techniques, the Italian black hats that broke into the Solaris system used one of these transition methods to configure communication with an IPv6 capable IRC server:

1. Dual-Stack hosts (running both an IPv4 and IPv6 stack at the same time)
2. Protocol Translation[24]
3. 6over4[25]
4. 6to4[26]
5. IPv6 Tunnel Broker[27]
6. Teredo (aka Shipworm)[28] (currently not a standard)

3.3.1 IPv6 tunnelled packets

Due to the ubiquitous nature of IPv4 on the internet, tunnelling has become a requirement, Figure 1 shows what an encapsulated packet looks like. Clients can create IPv6 tunnels over IPv4 from their individual machine or gateway connected directly to an IPv6 backbone, once the tunnel is established IPv6 communication flows over the IPv4 Internet seamlessly.

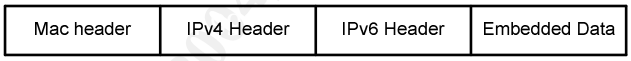


Figure 1 - IPv6 in IPv6 encapsulation

6over4 uses a dual IPv4/IPv6 stack host to communicate over an IPv4 multicast domain by encapsulating IPv6 packets within IPv4. 6to4 also uses a dual IPv4/IPv6 stack to encapsulate IPv6 packets as the traverse the IPv4 network, but uses a specially assigned address prefix that embeds your IPv4 address in the IPv6 address to make it easier to locate the endpoint of the tunnel.

Another very common method is via a Tunnel Broker. There are many Tunnel Brokers around the world, some of which provide a free service, they provide a portal into the native IPv6 network infrastructure on the Internet.

3.3.2 Tunnel Brokers

Tunnel Brokers provide an easy to configure solution for stand-alone clients or isolated networks to connect to an IPv6 backbone. The idea is to provide clients a stable connection and if required, a permanent IPv6 address that can be used across connections even if your IPv4 address changes, for example if you use a dial up internet

---

[22] Next Generation Transition (ngtrans) working group home page http://6bone.net/ngtrans/
[23] IPv6 Operations (v6ops) working group has taken on from the ngtrans efforts http://6bone.net/v6ops/
[24] RFC2766 "Network Address Translation - Protocol Translation (NAT-PT)"
[25] RFC2529 "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels"
[26] RFC3056 "Connection of IPv6 Domains via IPv4 Clouds
[27] RFC3053 "IPv6 Tunnel Broker"
[28] Microsoft XP and above support this mechanism, further details can be found at http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/teredo.mspx

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

connection which nearly always gives you a new address each time you connect. The basic Tunnel Broker architecture is shown in Figure 2, client configuration can be manual or scripted depending on the tunnel broker.

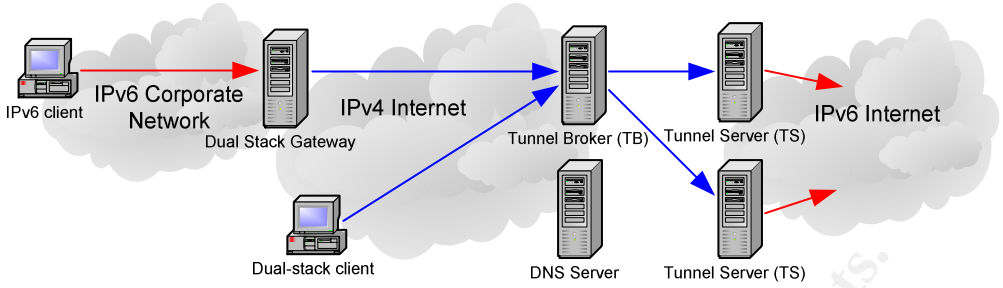


Figure 2 - Tunnel Broker configuration

Clients are dual-stack hosts that can be an individual system or a gateway for an IPv6 network. Clients make the initial connection to The Tunnel Broker (TB) who manages the tunnels. Once authorised, the TB instructs the Tunnel Server (TS) to create the endpoint on the server side, thus the actual tunnel is between the client system and the TS. The DNS server provides an IPv6 PTR record for clients. An example of a free Tunnel Broker is freenet6.org[29], they provide a small tool to configure the IPv6 tunnel removing the need for manual configuration.

3.3.3 IPv6 over IPv4 (6over4)

6over4 utilises an IPv4 multicast domain as an IPv6 virtual local link. This allows isolated IPv6 hosts not directly connected to an IPv6 router to become fully functional IPv6 host. Solaris 8 supports this type of tunnel also detailed in Volume 3 of the "System Administration Guide for Solaris 8" on page 364. The abridged steps are:

1. create the file `/etc/hostname6.ip.tunn`
   a. add the tunnel source and tunnel destination addresses to the file `tsrc` *`IPv4-source-addr`* `tdst` *`IPv4-destination-addr`* `up`
   b. optionally add a logical interface for the source and destination IPv6 address `addif` *`IPv6-source-address IPv6-destination-address`* `up`
2. The system must be rebooted to configure the tunnels
3. For bidirectional communication these steps must replicated on the other end of the tunnel

3.3.4 ipv6sun

The attacker uses a 6over4 tunnel to create the IPv6 connection, after the initial compromise the attacker downloads several files from an ftp server `62.211.66.16`, one of which was a script named `ipv6sun`, this is an IPv6 tunnel[30] setup script:

```
#!/bin/sh.
unset HISTFILE.
clear.
echo "Inserisci il tuo ipv4"; read ipv4tuo; echo "..Okz".
echo "Inserisci l'ipv4 del TunnelBroker"; read ipv4server; echo "..Okz".
echo "tsrc ipv4tuo tdst ipv4server up" > /etc/hostname6.ip.tun0.
echo "".
echo "Inserisci il tuo IPV6"; read ipv6tuo; echo "..Okz".
echo "Inserisci l'IPV6 dell'IRCServer"; read ipv6server; echo "..Okz".
echo "addif ipv6tuo ipv6server up" >> /etc/hostname6.ip.tun0.
echo "".
echo "TermiNateD =)".
echo "maphia-Groups r0x again".
```

[29] home page www.freenet6.org
[30] Within Ethereal the script can be obtained from the capture file `day1.log`, by running a "Follow TCP Stream" from position number 996.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The attacker appears to speak Italian hence, some parts of the script are cryptic. Looking at the script in detail the lines in bold show the purpose is to execute the two main steps for creating a 6over4 tunnel as detailed in the previous section. We will see how quick the attacker created the tunnel in a later section.

3.4 IPv6 packet Structure

Before we look at an IPv6 trace we need to understand more about its construction, the header is of a fixed size and has been designed with the minimum requirements, relegating any additional features to extension headers. Extension headers are analogous to IPv4's options, but provide a lot more flexible and expandable solution. There are currently six extension headers defined or referenced in RFC2460:

- *Hop-by-Hop Options;*
  This option carries information that must be examined by every node along a packets delivery path
- *Routing (Type 0);*
  This extension header is similar to the IPv4 Loose Source and Record Route options, it defines a list of one or more intermediate nodes to be "Visited" on the way to a packet's destination.
- *Fragment;*
  As previously mentioned, with the improvements in MTU discovery fragmentation is unnecessary. The option is still available but is only implemented by the source of the packet, no routers along the path can fragment an IPv6 packet.
- *Destination Options;*
  These Extension headers only contain information for the destination host.
- *Authentication[31]; and*
- *Encapsulating Security Payload[32].*
  These IPSEC[33] Extension headers offer exactly the same features and functionality as for IPv4, they where initially defined for IPv6 but back ported to provide security and extend its lifetime.

3.4.1 The IPv6 header

Figure 3 shows the IPv6 packet compared to IPv4.

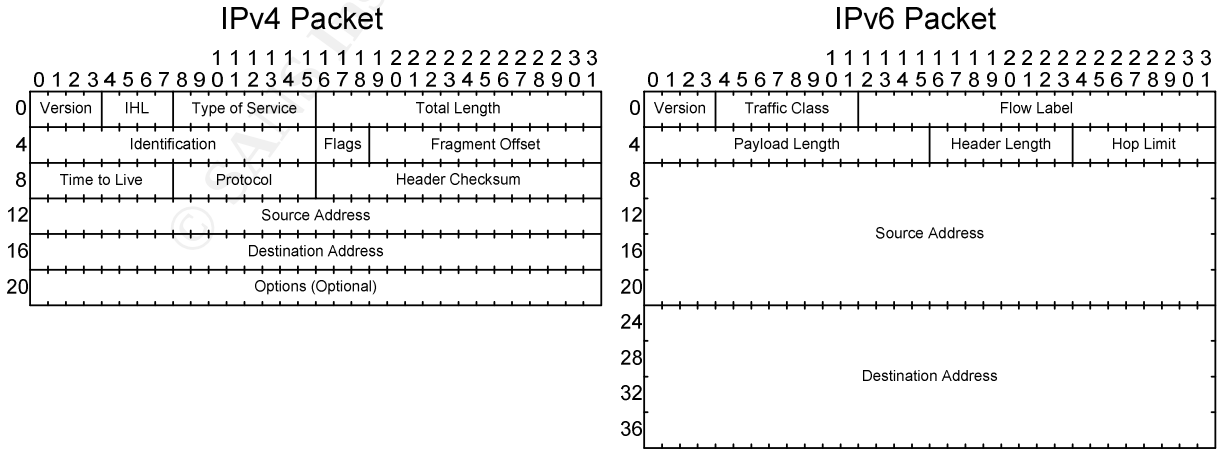


Figure 3 - IPv4 and IPv6 packet structure

[31] RFC2402 "IP Authentication Header"
[32] RFC2406 "IP Encapsulating Security Payload (ESP)"
[33] RFC2401 "Security Architecture for the Internet Protocol (IPSEC)"

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

It is easy to see there are striking differences with the construction of each. Not only is the IPv6 header significantly larger than an IPv4 packet without options, it also has far less fields included in the main packet. 3.5.2 IPv6 header component shows a detailed description of the IPv6 header components taken from our network trace.

3.5 IPv6 ICMP packet encapsulated in IPv4

To provide a concise explanation of the IPv6 packets construction this section breaks down a real world IPv6 packet encapsulated in IPv4 from the `day3.log` file. This is displayed by using the tcpdump command[34]:

```
# /usr/sbin/tcpdump -nnXvve -r day3.log  -c 1 'ip[9] = 41'

09:59:36.060504 0:7:ec:b2:d0:a 8:0:20:d1:76:19 0800 106: 163.162.170.173 > 192.168.100.28:
ipv6 72 (ttl 11, id 30290, len 92, bad cksum 3ac2!)
0x0000   4500 005c 7652 0000 0b29 3ac2 a3a2 aaad        E..\vR...):.....
0x0010   c0a8 641c 6000 0000 0020 0001 fe80 0000        ..d.`...........
0x0020   0000 0000 0206 5bff fe04 5e95 ff02 0000        ......[...^.....
0x0030   0000 0000 0000 0001 ff00 5d0f 3a00 0100        ..........]:....
0x0040   0502 0000 8300 0d64 0000 0000 ff02 0000        .......d........
0x0050   0000 0000 0000 0001 ff00 5d0f                  ..........].
```

This packet effectively looks like this:

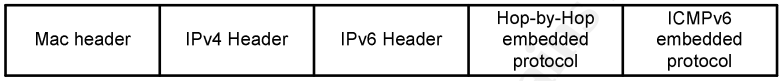


Figure 4 - Encapsulated ICMPv6 packet

3.5.1 IPv4 header component[35]

Except for carrying a protocol 41 packet (IPv6), this is a standard IPv4 header.

```
0x0000   4500 005c 7652 0000 0b29 3ac2 a3a2 aaad        E..\vR...):.....
0x0010   c0a8 641c 6000 0000 0020 0001 fe80 0000        ..d.`...........
```

Data	Description
4	IP version (4)
5	Header length for version 4 packet. (In this case, a typical 20-byte size indicating it does not contain any IPv4 options.)
00	Type of Service
005c	Total Length (`0x5c = 92` therefore the total packet length is 92 bytes)
7652	IP ID (`0x7652 = 30290`)
0000	Fragment flags and offset, this packet is not part of a fragment chain hence there is no requirement to have an offset or the more fragments flag set. Also in this case the don't fragment flag is not set.
0b	Time To Live of packet (`0xb = 11`)
29	Embedded protocol (`0x29 = 41 = IPv6` embedded protocol)
3ac2	header checksum (this seems to be corrupted, probably due to the packet mangling when scrubbing the trace)
a3a2 aaad	Source IP address is that of the attacker `0xa3 = 163, 0xa2 = 162, 0xaa = 170, 0xad = 173`
c0a8 641c	Destination IP address is that of the sun host (honeypot) `0xc0 = 192, 0xa8 = 168, 0x64 = 100, 0x1c = 28`

3.5.2 IPv6 header component[36]

This is the IPv4 embedded protocol, in our case it is ipv6.

```
0x0010   c0a8 641c 6000 0000 0020 0001 fe80 0000        ..d.`...........
0x0020   0000 0000 0206 5bff fe04 5e95 ff02 0000        ......[...^.....
0x0030   0000 0000 0000 0001 ff00 5d0f 3a00 0100        ..........]:....
```

Data	Description
6	IP version (6)
00	Traffic Class
00000	Flow Label, which in our case has not been set by the source host.
0020	Payload length (`0x20 = 32` therefore the payload length will be 32 bytes long) The payload length only represents the data after the main IPv6 packet, all extension headers are part of this calculation, but not the IPv6 header itself. The IPv6 packet is always of a known size and therefore not necessary to be part of the length calculation. Providing this quantity of bits only allow for a maximum packet size of 64kb, this really is not considered a limitation, Ethernet probably the most widely deployed "link layer" protocol only provides a maximum MTU of 1.5kb

[34] A description of the command is in Appendix A – Tcpdump command description.
[35] IPv4 was introduced in RFC791
[36] IPv6 was introduced in RFC2460
[37] http://www.geocities.com/SiliconValley/Vista/8672/network/ethernet.html#A25

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

	(1500bytes[37]). The default MTU size for IPv6 on Ethernet is 1500bytes[38]. Any packets required to be larger than this can use an extension header named jumbogram extension header[39].
00	Next header field (IPv4 terminology - Embedded Protocol) 0x00 indicates a hop by hop option header follows this header
01	Hop Limit (IPv4 terminology - TTL) in this case it has only one hop left, when IPv6 is encapsulated in IPv4, the distance it travels through the tunnel is considered one hop, with the hop count being so low this could indicate the attacker is trying to traceroute or something similar to the target host.
Fe80 0000 0000 0000 0206 5bff fe04 5e95	IPv6 source address, the address looks like fe80:0000:0000:0000:0206:5bff:fe04:5e95 A proper text representation of this address would look like[40] fe80::206:5bff:fe04:5e95
Ff02 0000 0000 0000 0000 0001 ff00 5d0f	IPv6 destination address, the address looks like ff02:0000:00000:0000:0000:0001:ff00:5d0f A proper text representation of this address would look like ff02::1:ff00:5d0f

3.5.3 Hop-by-Hop option header[41]

One of many possible option headers, in this case it is a Hop-by-Hop option header

```
0x0030   0000 0000 0000 0001 ff00 5d0f 3a00 0100        ..........:.....
0x0040   0502 0000 8300 0d64 0000 0000 ff02 0000        .......d........
```

Data	Description
3a	Next header (0x3a = 58 = ipv6 icmp)
00	Header Extension length (Length is calculated in units of 8 bytes, not including the first 8 bytes) this is set to zero as the total length is 8 bytes.
0100	This is a PadN extension option, Pad1 and PadN options are the only ones defined in RFC2460
0502 0000	Router alert: MLD.

3.5.4 ICMPv6 embedded protocol[42]

The next of many possible embedded protocols, in this case it is icmpv6.

```
0x0040   0502 0000 8300 0d64 0000 0000 ff02 0000        .......d........
0x0050   0000 0000 0000 0001 ff00 5d0f                  ..........].
```

Data	Description
83	ICMP type code (0x83 = 131 = multicast listener report)
00	ICMP Code
0d64	Checksum (in this case it is correct)
0000 0000	maximum response delay
ff02 0000 0000 0000 0000 0001 ff00 5d0f	Multicast address IPv6 address is ff02:0000:0000:0000:0000:0001:ff00:5d0f But when typed is usually represented as ff02::1:ff00:5d0f

4 The example trace

The initial compromise of the honeypot is thoroughly examined within the papers on the main SOTM28 page, there was only one main question in the challenge on IPv6[43]:

> Following the attack, the attacker(s) enabled a unique protocol that one would not expect to find on an IPv4 network. Can you identify that protocol and why it was used?

I am sure the standing of unusual for IPv6 has changed significantly since this attack, one of the entrants[44] even argued wether IPv6 was a usual protocol at all. For bonus points there where two additional questions on IPv6

> What are the implications of using the unusual IP protocol to the Intrusion Detection industry?
>
> What tools exist that can decode this protocol?

[38] RFC2464 details the Transmission of IPv6 packets over Ethernet Networks.
[39] Jumbograms are described in RFC2675
[40] This technique described in RFC3513 section "2.2 Text Representation of Addresses"
[41] The Hop-by-hop option header was introduced with IPv6 in RFC2460
[42] The main ICMPv6 protocol is described in RFC2463
[43] SOTM Challenge 28 main page – http://www.honeynet.org/scans/scan28/
[44] This is the paper that argues the relevance of IPv6 – http://honeynet.org/scans/scan28/sol/2/index.html#questionb1

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4.1 The compromise

The honeypot was compromised via a "`dtspcd`" buffer overflow attack, where the attacker proceeded to download additional tools, a root kit, and patches. IRC communications seemed to be one of the main reasons for compromising the host, as there are many attempts at IRC communications IPv6 communications. IPv6 was setup and activated with the primary connection being with an IPv6 enabled IRC chat server. The attack details are within the answers to the challenge[45].

4.2 Related traffic

Figure 5 is a connection diagram detailing the hosts involved in the compromise and subsequent configuration of IPv6. Each connection is shown with a description of the traffic type, the line number and time of connection as displayed in Ethereal[46]. Connections are from the `day3.log` binary log file unless mentioned otherwise.

Connections are displayed in two different colours, red (IPv6) and blue (IPv4). The IPv6 connections we will examine more closely while for completeness the IPv4 connections are detailed.

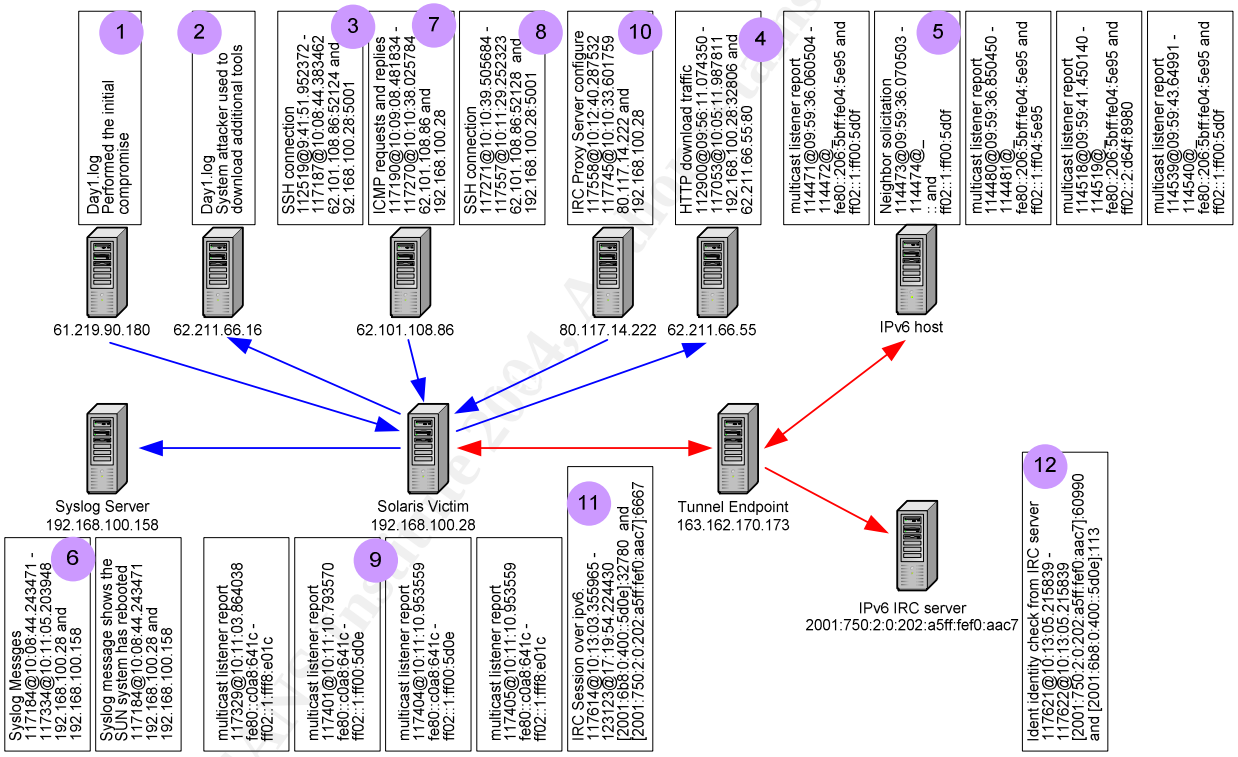


Figure 5 - Connection Diagram of relevant hosts

4.3 Traffic Timeline

Figure 6 shows the timeline of events the attacker used to configure the target host for IPv6 communication. Take note, the time scale varies throughout the diagram.

[45] The official SOTM28 answer is at http://www.honeynet.org/scans/scan28/sol/official/index.html
[46] Ethereal can be found at http://www.ethereal.com/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

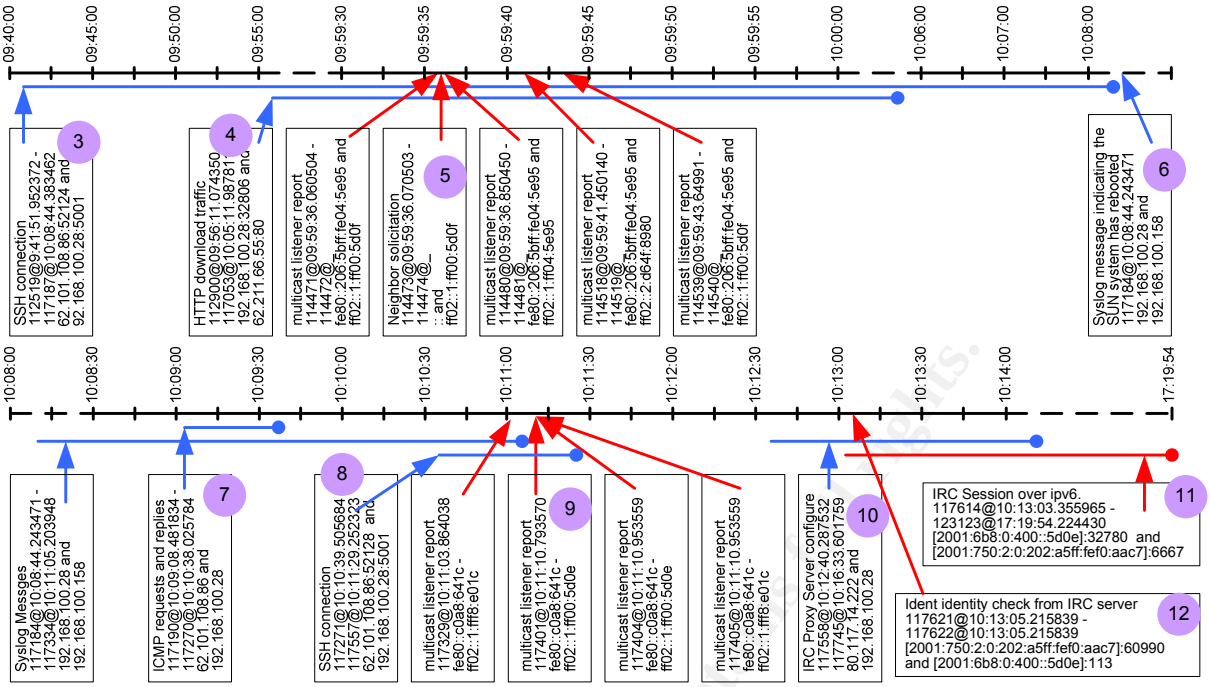


Figure 6 - Timeline of relevant packet, shown by time as displayed in Ethereal

A brief description of each event numbered in the previous two figures follows, events 1 and 2 are from the `day1.log` binary file and mentioned in Figure 5

From the `day1.log` log file

1. *Honeypot compromised*
   From host `61.219.90.180` the attacker used a dtspcd buffer overflow to compromise the honeypot.
2. *Tools download*
   Tools downloaded from `62.211.66.16` include a file named "`ipv6sun`" used to create the 6over4 IPv6 tunnel. (Setting up an IPv6 tunnel could have been the objective of the attacker from the start, even though it is configured a lot later)

From the `day3.log` log file

3. *SSH connection*
   Through a SSH connection from host `62.101.108.86` while downloading an additional file the attacker configures the IPv6 tunnel using the previously downloaded script. The ICMPv6 packets directed to the honeypot presumably from the attacker trying to test the IPv6 stack evidence this. After the HTTP download is completed there is around 3.5 minutes where I suspect the attacker configures the additional file and reboots the system, disconnecting the SSH session.
4. *HTTP traffic, downloading of software?*
   A new file named `psy.tar` is downloaded from the web server `62.211.66.55`, it contains `psy-bnc`, which is an IPv6 compatible IRC server/client as described in the official write-up by Raul Garcia for SOTM28[47].
5. *Five unusual ICMP packets*

[47] The official SOTM28 answer is at http://www.honeynet.org/scans/scan28/sol/official/index.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

While the attacker is downloading the `psy.tar` file, a series of ICMPv6 packets with an IPv4 source address of `163.162.170.173` is seen. This address appears to be the tunnel endpoint. Performing a PTR lookup on the address reveals:

```
173.170.162.163.in-addr.arpa. 3600 IN   PTR     ts.ipv6.tilab.com.
```

The '`ts`' in this FQDN presumably indicates Tunnel Server, which leads me to believe that the host generating these ICMPv6 packets could be the same as `62.101.108.86` with a dual stack installed and its own tunnel providing IPv6 connectivity. These could be an attempt by the attacker to verify if the IPv6 stack is working on the Solaris system, we know it needs a reboot before the IPv6 stack is initialised for communications.

6. *Syslog messages*
   It is uncertain if the attacker forgot they had to reboot the box before IPv6 connectivity would work, after configuring the new tool and a few failed attempts to illicit a positive response from the IPv6 stack the attacker restarts the system. Details are in the syslog messages sent to a central syslog server.
7. *IPv4 ICMP messages*
   The attacker needs to know when the system has completed the rebooted so they use IPv4 ICMP requests from `62.101.108.86` until the host responds again, at which point we then see ICMP echo replies indicating it has returned.
8. *SSH connection*
   Once the system is up again there is a new SSH connection that lasts for less than a minute from host `62.101.108.86`, the attacker uses this connection to test IPv6 connectivity and then disconnects.
9. *Multicast listener report*
   IPv6 ICMP packets seen leaving the host.
10. *IRC Proxy configuration*
    Just over a minute after the close of the second SSH connection there is a connection from `80.117.14.222`, previously throughout the `day3.log` file this host has made IPv4 IRC connections, here the host configures the tool `psy-bnc`, during this connection we see traffic like this (edited for brevity):

```
attack reqst =>       ADDSERVER 2001:750:2:0:202:a5ff:fef0:aac7:6667
server respn =>       Server 2001:750:2:0:202:a5ff:fef0:aac7 port 6667 (password: None)
                      added.
server respn =>       trying 2001:750:2:0:202:a5ff:fef0:aac7 port 6667
server respn =>       connected to 2001:750:2:0:202:a5ff:fef0:aac7:6667
server respn =>       :irc6.edisontel.it 001 `OwnZ`` :Welcome to the Internet Relay Network
                      `OwnZ``!~ahaa@host222-14.pool80117.interbusiness.it
attack reqst =>       SETAWAY -OwnZ-
server respn =>       :irc6.edisontel.it 002 `OwnZ`` :Your host is irc6.edisontel.it,
                      running version 2.10.3p3+hemp
```

11. *Internet Relay Chat Session over IPv6*
    At the same time the attackers tool announces it is `trying 2001:750:2:0:202:a5ff:fef0:aac7 port 6667` we see an IPv6 connection to the IRC server initiated from the honeypot.
12. *Ident check*
    The Internet Relay Chat server tries to make an ident call back to the honeypot in an attempt to identify the user that is connecting to the server. This check fails with an immediate reset indicating that the service is not listening on the honeypot.

5 IPv6 addressing

We have been exposed to IPv6 addresses like `2001:750:2:0:202:a5ff:fef0:aac7` already in this paper, the increased address range from 32bits to 128bits is one of the most obvious and well advertised changes in the new protocol. This section will talk about

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

the new addressing scheme, the allocation of addresses and provide a clue to deciphering these long numbers, along the way we will see how smart the new protocol is.

IPv4 addresses are made up of four distinct segments separated by a period (`.`) commonly known as "dotted decimal notation" each segment contains a decimal representation of an 8-bit value. An IPv6 address in made up of 8 segments separated by a colon (`:`) each containing a hexadecimal representation of a 16bit value. Here is an example address.

```
2001:06b8:0000:0400:0000:0000:0000:5d0e
```

IPv6 addresses like this are difficult to represent in text so there are two transformations we can make, the first is to remove any leading zeros from within each address segment.

```
2001:6b8:0:400:0:0:0:5d0e
```

Lastly, due to the large size of this address space, it is quite plausible to have a string of segments containing zeros, these can be reduced by representing this string with a double colon (`::`) though this can only be done once within any address.

```
2001:6b8:0:400::5d0e
```

There are three basic types of addresses available[48]:

- *unicast*
  This is a unique address assigned to a single interface on a host as in IPv4. There are several types of unicast addresses including global unicast, site-local unicast, and link-local unicast.
- *anycast*
  This is a unique address type assigned to multiple interfaces, ideally on different hosts. A packet directed to an anycast address is delivered to the closest host as identified by the routing protocols measure of distance, and this host only.
- *multicast*
  Another familiar address type is the multicast address. Any packet directed to a multicast address is delivered to all nodes listening on that address. This functionality replaces the broadcast address type found in IPv4, hosts register to special multicast groups to receive local broadcasts.

5.1 IPv6 address

Similar to the way IPv4 was separated into classes like A, B and C class identified by their high order bits, IPv6 address types are also identified by their high order bits[49]:

Table 1 - Break Up of the IPv6 Address Range

Unspecified	`00...0 (128 bits)`	`::/128`
Loopback	`00...1 (128 bits)`	`::1/128`
Multicast	`11111111`	`FF00::/8`
Link-local unicast	`1111111010`	`FE80::/10`
Site-local unicast	`1111111011`	`FEC0::/10`
Global unicast	(everything else)	

The unspecified address `::` is equivalent to IPv4's `0.0.0.0` while the loopback `127.0.0.1` address is described as `::1` in IPv6 format. Similar to IPv4, IPv6 network masks use CIDR [50] (Classless Inter-Domain Routing) notation, however, unlike IPv4 any given interface is expected to have more than one address, RFC3513 stipulates the addresses a host requires, each is briefly discuss in the following sections:

- A Link-Local Address for each interface;
- Any additional Unicast and Anycast addresses that have been configured for the node's interfaces either manually or automatically;

[48] RFC3513 "Internet Protocol Version 6 (IPv6) Addressing Architecture"
[49] internet protocol version 6 address space http://www.iana.org/assignments/ipv6-address-space
[50] This technique is detailed in RFC1519 "Classless Inter-Domain Routing (CIDR)"

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- The Loopback address (This has already been mentioned);
- All-Nodes Multicast Addresses;
- A Solicited-Node Multicast Address for each of its unicast and anycast addresses; and
- Multicast Addresses of all other groups to which the node belongs.

Address assignment is easy with the use of automatic address configuration techniques, there are techniques that give each host the ability to discover each other and keep track of the changing conditions in the network. DHCP though still available as DHCPv6[51] [52] is not required for dynamic addressing, dynamically assigned address are built into the way IPv6 operates, network routers are designed to play a greater role in the process of address assignment, they can advertise the network prefix for clients to use when configuring their addresses.

5.2 Pv6 addresses seen in the network trace

The IPv6 addresses seen in the trace are form the `day3.log` file, there are of a few different addresses that are detailed in Table 2. Not all IPv6 connections are in the table, only connections that are either interesting or observed for the first time seen, the Ethereal line number is provided to ease location. The next few sections detail the various IPv6 address types, not all are observed in our trace but the IDS analyst should understand them, as they are sure to be seen in other traces.

As previously described, the IPv6 packets are encapsulated in IPv4 so the source IPv4 address is also included with the Ethernet MAC address for completeness. The IPv4 addresses provide a clue to the tunnel ends, there is no native IPv6 traffic observed. This host is configured as a standalone IPv6 client connecting into a Tunnel Server, the red connections in Figure 5 show how these packets are routed.

Table 2 - day3.log IPv6 addresses

line no.	IPv6 address	IPv4 address	MAC address	protocol
114471	fe80::206:5bff:fe04:5e95	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
114471	ff02::1:ff00:5d0f	192.168.100.28	08:00:20:d1:76:19	ICMPv6
114473	::	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
114473	ff02::1:ff00:5d0f	192.168.100.28	08:00:20:d1:76:19	ICMPv6
114480	fe80::206:5bff:fe04:5e95	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
114480	ff02::1:ff04:5e95	192.168.100.28	08:00:20:d1:76:19	ICMPv6
114518	fe80::206:5bff:fe04:5e95	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
114518	ff02::2:d64f:8980	192.168.100.28	08:00:20:d1:76:19	ICMPv6
114539	fe80::206:5bff:fe04:5e95	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
114539	ff02::1:ff00:5d0f	192.168.100.28	08:00:20:d1:76:19	ICMPv6
The Honeypot was rebooted at this point before we see these next addresses				
117401	fe80::c0a8:641c	192.168.100.28	08:00:20:d1:76:19	ICMPv6
117401	ff02::1:ff00:5d0e	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
117404	fe80::c0a8:641c	192.168.100.28	08:00:20:d1:76:19	ICMPv6
117404	ff02::1:ff00:5d0e	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
117405	fe80::c0a8:641c	192.168.100.28	08:00:20:d1:76:19	ICMPv6
117405	ff02::1:fff8:e01c	163.162.170.173	00:07:e3c:b2:d0:0a	ICMPv6
117614	2001:6b8:0:400::5d0e	192.168.100.28	08:00:20:d1:76:19	TCP
117614	2001:750:2:0:202:a5ff:fef0:aac7	163.162.170.173	00:07:e3c:b2:d0:0a	TCP
117621	2001:750:2:0:202:a5ff:fef0:aac7	163.162.170.173	00:07:e3c:b2:d0:0a	TCP
117622	2001:6b8:0:400::5d0e	192.168.100.28	08:00:20:d1:76:19	TCP

5.3 Link Local Addresses[53]

One of the first addresses seen from the attacker before the system reboot is `fe80::206:5bff:fe04:5e95`. This is a link local address identifiable as it falls into the `fe80::/10` address range and is followed by a 64-bit Interface Identifier. This address type is assigned to an interface at boot up so it has an address for communication straight

[51] Dynamic Host Configuration Protocol for IPv6 (DHCPv6) is documented in RFC3315
[52] Reference site for DHCPv6 – http://www.dhcpv6.org/
[53] Main details are in RFC2463 "IPv6 Stateless Address Autoconfiguration"

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

away. They are only relevant to the local network (equivalent to the IPv4 broadcast domain) as a means of local communication between hosts. They are used in the neighbour discovery process, and must not be forwarded by an IPv6 router.

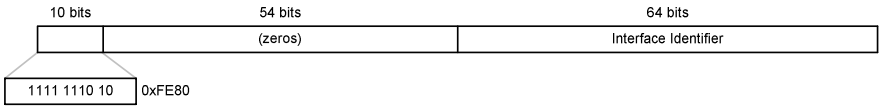


Figure 7 - Link Local Address Structure

The first 10 bits are always `fe80` hex or `1111 1110 10` binary, the middle 54 bits are set to zero, and the last 64bits make the Interface Identifier which in our example is `206:5bff:fe04:5e95`. This is a `EUI-64` formatted Interface Identifier explained further in section 5.6. It is constructed using the connecting interface MAC (Media Access Control) address that should be unique as all MAC addresses by nature should be unique across all computers around the world. As uniqueness is not a guarantee, there are processes in place to verify uniqueness as described in RFC2461 (Neighbour Discovery in IPv6).

5.4 Site Local Addresses[54]

There are no Site Local addresses in our trace but they fall into the address range `feC0::/10` also followed by a `EUI-64` formatted interface ID. These address types are analogous to private IPv4 addresses as `10.0.0.0/8` defined in RFC1918, and must not be forwarded beyond the local site by an IPv6 router. This address type is for sites that have multiple internal networks.

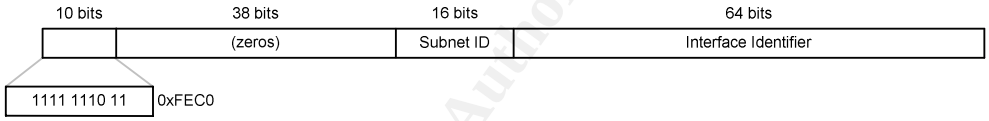


Figure 8 - Site Local Address Structure

5.5 Global Unicast addresses

Global Unicast addresses also have an Interface Identifier, these can be created dynamically by the host as a `EUI-64` formatted address, specifically assigned via DHCPv6 or manually configured. The observed address `2001:6b8:0:400::5d0e` has an interface identifier of `5d0e` with the rest of the interface identifier being set to zero.

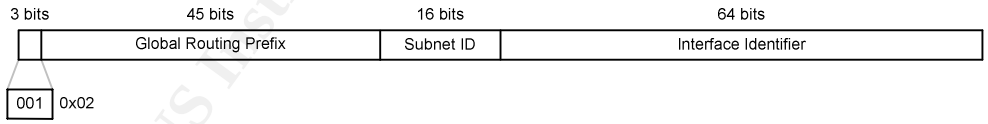


Figure 9 - Link Local Address Structure

Even though Table 1 suggests Global Unicast addresses cover all other non specified address blocks, the only range formally assigned start with a `001` binary prefix or `2000::/3`[55] [56]. All other address blocks are to the most part unassigned. Within this range IANA (Internet Assigned Numbers Authority) have allocated blocks of addresses to the Regional Internet Registries (RIR)[57]:

Table 3 - IANA allocated address blocks

Prefix	Binary representation	Regional Registry	Date Assigned
2001:0000::/23	0000 000X XXXX X	IANA	Jul 99
2001:0200::/23	**0000 001X XXXX X**	**APNIC**	**Jul 99**

[54] Main details are in RFC2463 "IPv6 Stateless Address Autoconfiguration"
[55] Which is further defined in RFC3587 "IPv6 Global UnicastAddress Format"
[56] IANA's IPv6 Address Allocation and Assignment Policy:
http://www.iana.org/ipaddress/ipv6-allocation-policy-26jun02
[57] For the latest IPv6 top level aggregation identifier assignments: http://www.iana.org/assignments/ipv6-tla-assignments

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2001:0400::/23	0000 010X XXXX X	ARIN	Jul 99
2001:0600::/23	**0000 011X XXXX X**	**RIPE NCC**	**Jul 99**
2001:0800::/23	0000 100X XXXX X	RIPE NCC	May 02
2001:0A00::/23	0000 101X XXXX X	RIPE NCC	Nov 02
2001:0C00::/23	0000 110X XXXX X	APNIC	May 02
2001:0E00::/23	0000 111X XXXX X	APNIC	Jan 03
2001:1000::/23	0001 000X XXXX X	(future assignment)	
2001:1200::/23	0001 001X XXXX X	LACNIC	Nov 02
2001:1400::/23	0001 010X XXXX X	RIPE NCC	Feb 03
2001:1600::/23	0001 011X XXXX X	RIPE NCC	Jul 03
2001:1800::/23	0001 100X XXXX X	ARIN	Apr 03
2001:1A00::/23	0001 101X XXXX X	RIPE NCC	Jan 04
and continues until			
2001:FE00::/23	1111 111X XXXX X	(future assignment)	

The addresses `2001:6b8:0:400::5d0e` and `2001:750:2:0:202:a5ff:fef0:aac7` used in the IRC communication are global unicast addresses within the range assigned to the RIPE NCC Regional Internet Registry (shown in blue in Table 3). This is the first place to investigate these addresses further. The Regional Registries inturn allocate these ranges either to the National Internet Registries, Local Internet Registries or directly to Internet Service Providers[58].

The regional registries are:

ARIN (American Registry for Internet Numbers) – North America and Sub-Sahara Africa
Home page http://www.arin.net/
whois search http://www.arin.net/whois/index.html

LACNIC (Regional Latin-American and Caribbean IP Address Registry) – Latin America and some Caribbean Islands
Home page http://lacnic.net/en/index.html
whois search http://lacnic.net/cgi-bin/lacnic/whois

RIPE NCC (Réseaux IP Européens) – Europe, the Middle East, Central Asia, and African countries located north of the equator
Home page http://www.ripe.net/
whois search http://www.ripe.net/db/whois/whois.html

APNIC (Asia Pacific Network Information Centre) – Asia/Pacific Region
Home page http://www.apnic.net/
whois search http://www.apnic.net/apnic-bin/whois.pl

As an example of the next level of delegation, we can see how the APNIC has assigned the range shown in bold in Table 3 at http://ftp.apnic.net/stats/apnic/delegated-apnic-latest. As an example of the content, we can see further subdivision of the address block `2001:0200::/23` as follows[59]:

Table 4 - APNIC assigned address blocks

apnic|JP|ipv6|2001:200::|32|19990813|allocated
apnic|SG|ipv6|2001:208::|32|19990827|allocated
apnic|AU|ipv6|2001:210::|35|19990916|allocated
apnic|JP|ipv6|2001:218::|32|19990922|allocated
apnic|KR|ipv6|2001:220::|32|19991006|allocated
and continues
apnic|JP|ipv6|2001:3d8::|32|20020830|allocated
apnic|JP|ipv6|2001:3e0::|32|20020524|allocated
apnic|JP|ipv6|2001:3e8::|32|20020609|allocated
apnic|JP|ipv6|2001:3f0::|32|20020704|allocated
apnic|CN|ipv6|2001:3f8::|32|20020704|allocated

[58] RFC3177 provides recommendations for regional registries when allocating IPv6 address.
[59] For a description of the format used in these files visit http://www.apnic.net/db/rir-stats-format.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

It is just as easy to perform a "whois" lookup at each registry to find more details on a particular IPv6 address as it is with IPv4 addresses, we will see an example of this later.

5.6 Interface Identifier (EUI-64)

Figure 10 shows how the EUI-64 Interface ID used by many address types, it is constructed starting with the original MAC address[60] and converted into the Interface Identifier, our example is the link local address fe80::**206:5bff:fe04:5e95** seen is the trace:

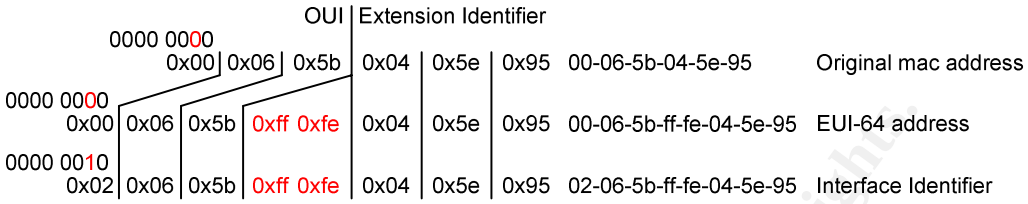


Figure 10 - Converting a MAC address to an IPv6 Interface Identifier

A standard MAC-48 address has two components, an OUI (organisationally Unique Identifier) component and the Extension Identifier. The OUI is the first half of the MAC address and assigned as block to a requesting organisation. The Extension Identifier is determined by the organisation assigned the OUI. 0xfffe is inserted between the two components to convert the MAC-48 address into a EUI-64[61] (Extended Unique Identifier). The 0xfffe is specially used to convert a MAC-48 address into an EUI-64 address allowing the MAC-48 address to be encapsulated within the EUI-64 address while maintaining it's uniqueness on the network. The EUI-64 address is then converted into an Interface Identifier by complementing the "Universal/Local" (U/L) bit that is the next-to-lowest ordered bit in the first octet of the EUI-64

This address is from a tunnelled connection but because the source host is using a link local address, we can return the Interface Identifier back to the host's original MAC address. Figure 10 shows the original MAC address is 00-06-5b-04-5e-95, looking up the address on the IEEE website reveals some information about the source host[62].

```
00-06-5B    (hex)            Dell Computer Corp.
00065B      (base 16)        Dell Computer Corp.
                             One Dell Way
                             Round Rock TX 78682
                             UNITED STATES
```

For both security and privacy reasons being able to determine this level of information on a host could be considered a big issue when attached global unicast or site-local addresses, for link-local addresses pose less of a risk as you are expected to be able to identify the MAC address on you local network. The EUI-64 address is designed to be globally unique making the host traceable across multiple connections as it travels from network to network with the prefix being the only change. This is a known issue that has been addressed in RFC3041, this defines a method to create a unique address from the hosts EUI-64 address that changes at regular intervals making it extremely difficult to track a user based on their IP address, but still maintains uniqueness on the network.

5.7 Global Anycast addresses

A global anycast address is assigned from the same pool of addresses as the global unicast address pool making them indistinguishable from a unicast address. A packet directed to an anycast address is delivered to the closest node with that address as determined by the routing protocols unit of measure. The anycast address though

[60] More information on this process can be obtained from RFC2464
[61] More information on EUI-64 – http://standards.ieee.org/regauth/oui/tutorials/EUI64.html
[62] This search was done from http://standards.ieee.org/regauth/oui/index.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

assigned to multiple nodes cannot be the source of a connection presumably because you cannot guarantee the reply will return to the originating host.

5.8 Multicast addresses

Multicast addresses are designed to perform a similar function they do in IPv4, multicast addresses are what make IPv6 extremely flexible, they form part of the core functionality that lessens load on hosts. IPv6 uses multicast groups to perform the same functionality used in IPv4 broadcasts. The idea is each group has a specific purpose that hosts can join or leave as desired, hosts only need to process messages it requires, thus reducing unnecessary broadcast traffic that in the end most of the hosts do not need to process.

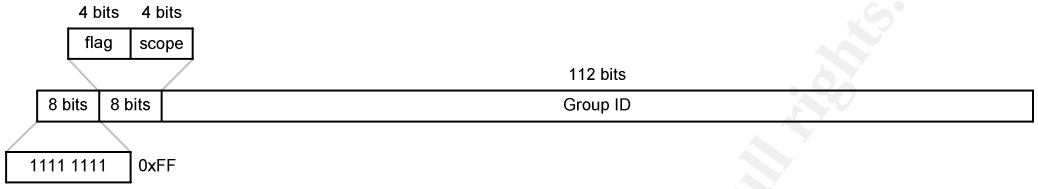


Figure 11 - Multicast Address

A multicast address is identified by `FF` hex or `1111 1111` binary at its head. Only the low order bit is used for the multicast flag, the rest are reserved and must always be set to zero. A `0` (zero) in the low order bit indicates a permanent "well known" address, while a `1` (one) indicates a temporary address. The scope field indicates the coverage this multicast group has, Table 5 shows the values the scope field can have[63].

Table 5 - Multicast Scope Values

value	Definition
0	reserved
1	interface-local scope
2	link-local scope
3	reserved
4	admin-local scope
5	site-local scope
6	(unassigned)
7	(unassigned)

value	Definition
8	organisation-local scope
9	(unassigned)
A	(unassigned)
B	(unassigned)
C	(unassigned)
D	(unassigned)
E	global scope
F	reserved

Link local and site local multicast addresses cover the same network area their corresponding unicast addresses do. For example, the link local multicast address only covers the same are as an IPv4 broadcast domain and should not be forwarded by an IPv6 router. There are many permanently assigned fixed scope multicast addresses[64] that provide a specific function on the network, some examples are:

Table 6 - Example of Permanent Multicast Addresses

IPv6 address	Address description
ff02::1	All Nodes Address (all nodes on the local link will process these messages)
ff02::2	All Routers Address (all routers within the local link will process these messages)
ff02::1:ffXX:XXXX	Solicited-Node Address
ff02::1:2	All DHCP agents (all DHCP agents on the local link will process these messages)
ff05::2	All Routers Address (all routers within the site will process these messages)
Ff05::1:3	All DHCP servers (all DHCP servers with the site will process these messages)

Section 5.1 mentioned IPv6 addresses hosts must have, some are multicast groups that a host must join to be considered a functioning IPv6 node, and these are:

- All-Nodes Multicast Addresses;
- A Solicited-Node Multicast Address for each of its unicast and anycast addresses; and
- Multicast Addresses of all other groups to which the node belongs.

[63] From RFC3513 page 14
[64] Internet protocol version 6 multicast addresses – http://www.iana.org/assignments/ipv6-multicast-addresses

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

We can now identify some of the addresses in our trace as Solicited-Node Address multicast packets, Table 7 shows the multicast addresses seen in our example trace that have been the destination for certain traffic.

Table 7 - Multicast Addresses in our Trace

IPv6 address	IPv4 address	Address type
ff02::1:ff00:5d0f	192.168.100.28	Solicited node address
ff02::1:ff04:5e95	192.168.100.28	Solicited node address
ff02::2:d64f:8980	192.168.100.28	All routers address
The Honeypot was rebooted at this point		
ff02::1:ff00:5d0e	163.162.170.173	Solicited node address
ff02::1:fff8:e01c	163.162.170.173	Solicited node address

Solicited-Node addresses are determined from the nodes unicast and anycast addresses by taking the low order 24 bits of an address and appending those to an `ff02::1:FFXX:XXXX` prefix. These addresses are created for each unicast or anycast address assigned to the node, they are used in neighbour solicitation messages to help with neighbour discovery. Taking a trace address of `ff02::1:ff00:5d0f` we can see the last 24 bits are `005d0f` Hex.

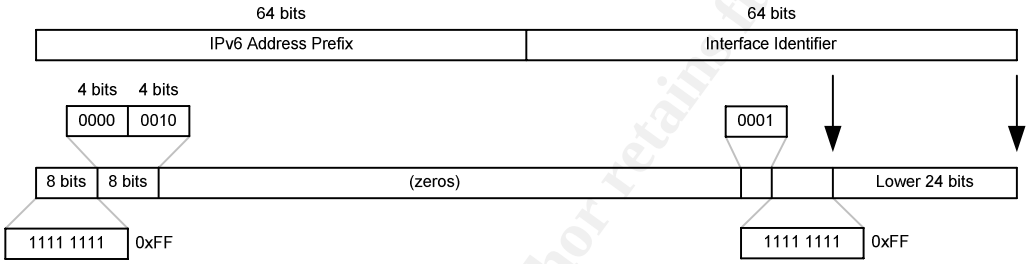


Figure 12 - Solicited-Node Multicast Address

5.9 Other address types

RFC3513 defines a couple of other address types, these are used as part of the migration process and include the "IPv4 compatible IPv6 address" which is used to dynamically tunnel IPv6 packets over an IPv4 routing infrastructure.

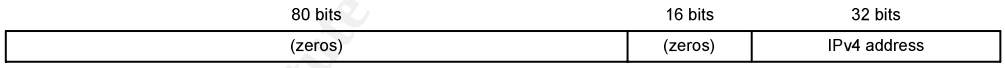


Figure 13 - IPv4 Compatible IPv6 Address

There is also the "IPv4 mapped IPv6 address" used to represent an IPv4 address as an IPv6 address.

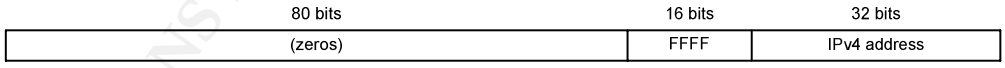


Figure 14 - IPv4 mapped IPv6 Address

RFC2529 defines another address type used in 6over4 communications; it is a link local address with an embedded IPv4 address. The non-global scope IPv6 addresses you see leaving the honeypot after it is rebooted are `fe80::c0a8:641c` are of this type, the `c0a8:641c` component of this address is hex for `192.168.100.28`.

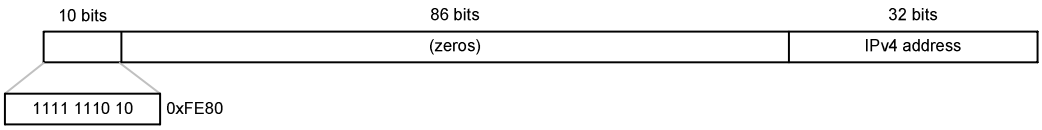


Figure 15 - IPv6 Link-local address for and IPv4 virtual interface

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

5.10 Investigating IPv6 addresses

The IPv6 Address Oracle[65] removes some of the pain associated with identifying IPv6 address types, after entering your IPv6 address it gives you:

```
detailed information about the structure of the addresses and
references to the standards documents that define the address.
```

The same methods used when investigating the owner of an IPv4 address can be used in the IPv6 world, performing a "`whois`" on the address obtained by the honeypot during the tunnel configuration identifies the owner of the address block[66].

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html
inet6num:     2001:6b8::/48
netname:      IT-NGNET-20021111
descr:        Telecom Italia Lab S.p.A.
descr:        ngnet.it initiative
descr:        www.ngnet.it/e/index.php
country:      IT
admin-c:      NGN2-RIPE
tech-c:       NGN2-RIPE
notify:       ipv6@tilab.com
mnt-by:       CSELT-IPV6-MNT
mnt-lower:    CSELT-IPV6-MNT
status:       ASSIGNED
changed:      ipv6@tilab.com 20021111
source:       RIPE
role:         ngnet Initiative
address:      Telecom Italia Lab S.p.A.
address:      Via Reiss Romoli, 274}
address:      I-10148 Torino (Italy)
e-mail:       info@ngnet.it
trouble:      ************************************************
trouble:      *  For ABUSE mail to:             abuse@ngnet.it   *
trouble:      *  For generic INFO mail to:      info@ngnet.it    *
trouble:      *  For OPERATIONAL ISSUES mail to: ipv6@tilab.com  *
trouble:      ************************************************
admin-c:      FI144-RIPE
tech-c:       FI144-RIPE
tech-c:       MM6609-RIPE
tech-c:          IG1588-RIPE
notify:       ipv6@tilab.com
mnt-by:       CSELT-IPV6-MNT
changed:      ipv6@tilab.com 20040204
source:       RIPE
nic-hdl:      NGN2-RIPE
```

Another lookup on the IRC servers IPv6 address[67] shows the block is assigned to Edisontel S.p.A. also providing information to report abuse etc.

The UNIX based DIG utility can perform reverse lookups on IPv6 addresses:

```
$ dig e.0.d.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int ptr

; <<>> DiG 9.2.3 <<>> e.0.d.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int ptr
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24377
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; QUESTION SECTION:
;e.0.d.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int. IN PTR

;; ANSWER SECTION:
e.0.d.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int. 30 IN PTR
        abCdifgHigkLmnOpqRstuvWxyZ.abCdifgHigkLMnOpqRstUvWxyZ.la.
```

[65] Advanced Network Management Laboratory IPv6 Address Oracle http://steinbeck.ucs.indiana.edu:47401/

[66] http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=2001%3A6b8%3A0%3A400%3A%3A5d0e&do_search=Search

[67] http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=2001%3A750%3A2%3A%3A202%3Aa5ff%3Afef0%3Aaac7&do_search=Search

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
;; AUTHORITY SECTION:
0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int. 60 IN NS kenoby.ipv6.cselt.it.
0.0.4.0.0.0.0.0.8.b.6.0.1.0.0.2.ip6.int. 60 IN NS griselda.ipv6.tilab.com.

;; ADDITIONAL SECTION:
griselda.ipv6.tilab.com. 3334   IN      A       163.162.170.180

;; Query time: 990 msec
;; SERVER: 192.168.111.10#53(192.168.111.10)
;; WHEN: Mon Apr 19 23:10:07 2004
;; MSG SIZE  rcvd: 246
```

6 Conclusion

Through the Honeynet project, we have been privileged to witness first hand the tools and tactics used by the black hat community, this paper I hope has shown the importance for the IDS analyst to become familiar with this protocol through the presentation of an IPv6 based attack. Even if you do not run any IPv6 systems on you network, this attack has demonstrated the ease at which an attacker can configure a compromised system to support IPv6 and using some of the migration techniques, tunnel IPv6 traffic over the internet.

There is a lot that a system administrator could do to detect this type of activity, some of the things include:

1. Look for traffic that is non-standard on your network, if you do not expect to see protocol 41 on your network then you should be alarmed if you do.
2. Detect the reboot of the system
3. Patch the system adequately to avoid the initial compromise.
4. Profile traffic entering and leaving your network and/or hosts. An increase in traffic generated by an attacker who is making your host their new home should trigger alarm bells.
5. Don't rely on any single source of information to determine if you have a compromised host, syslog messages are a great source of information as was evidenced by the message indicating the machine had restarted, though this will only work if the attacker has not completely disabled the syslog daemon.

7 Appendix

7.1 Appendix A – Tcpdump command description

```
# /usr/sbin/tcpdump -nnXvve -r day3.log  -c 1 'ip[9] = 41'
```

Command description

`/usr/sbin/tcpdump` The actual executable that captures data from the network. The reason I use the full path is to ensure I am running the correct executable and not another that could be in my path, a paranoid though good habit to be in.

`-nn` This is a command switch that stops tcpdump from resolving the names of IP address, protocol and port numbers as it collects data. This saves processing power and time. Besides, in my opinion, it is easier to work with numbers. If I need the name, I will look it up via other means.

`X` Tells tcpdump to print the packet in hex as well as ASCII.

`vv` Print very verbose information.

`e` Display the link-level header

`-r day3.log` Read packets from the `day3.log` file

`-c 1` Display only one packet

`'ip[9] = 41'` Display only packets with a IP protocol equal to 41 (IPv6)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

7.2 Appendix B – References

Almost all the references are inline with the text; here are some other references used during the compilation of this paper:

http://www.spitzner.net/honeypots.html
Honeypots - Definitions and Value of Honeypots
By Lance Spitzner, 29 May, 2003

http://www.honeynet.org/scans/index.html
'Scan of the Month' challenge main page
Honeynet project team

http://www.honeynet.org/scans/scan28/sol/official/index.html
Official write-up for SOTM 28
Raul Garcia of the Mexico Honeynet Project May 30, 2003

http://docs.sun.com/
Sun Microsystems, Inc.
System Administration Guide, Volume 3, February 2000

http://www.networksorcery.com/enp/default0601.htm
The *RFC Sourcebook* is a comprehensive guide to the Request for Comments (RFCs) series of Internet standards and Internet protocols

http://www.rfc-editor.org/rfcsearch.html
All RFCs discussed throughout the paper can be locate through this RFC search engine

http://www.cisco.com/warp/public/732/abc/docs/abcipv6.pdf
The ABCs of IP Version 6
CISCO IOS Learning Services

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1 Attack host 10.10.10.186

As a practical requirement this detect was posted to the intrusions.org mailing list and is accessible at http://www.dshield.org/pipermail/intrusions/2004-May/008018.php. At the end of this detect I have included the top three questions and answers as posted on the mailing list.

1.1 Source of Trace

This detect was taken from files contained within the tar gzip file `2003.12.15.tgz` located at http://www.incidents.org/logs/raw/. This archive contains 14 separate binary log capture files named `2003.12.15.1` to `2003.12.15.14`.

There is a lot of activity recorded in these files, I have chosen to concentrate on traffic generated by the subject host `10.10.10.186`. There is pertinent information relating to this host in all the log files except the last `2003.12.15.14` so this discussion will cover the whole set of traces within this archive set. Among others this host is seen attacking `172.20.201.198`, these are the two systems I have chosen to write this detect on.

I constructed this diagram and specific information on each host using techniques as identified by Ian Martin[68], who inturn used ideas courtesy of Les Gordon[69] and Andre Cormier[70].

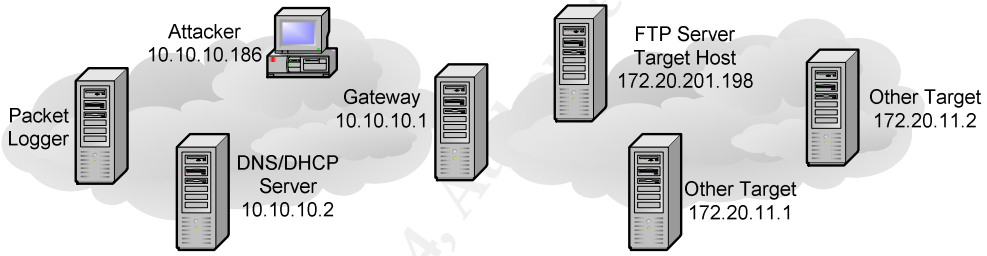


Figure 16 – Identified Network Layout

Details on the Attacking host:

Who	Attacker
IP	10.10.10.186
MAC	0:2:a5:b6:e2:e3
MAC owner	Compaq Computer Corporation
P0f	Linux 2.4/2.6 (up: 2 hrs)

Other players in the trace include:

Who	DNS/DHCP server
IP	10.10.10.2
MAC	0:50:56:40:0:64
MAC owner	VMWare
P0f	?

Who	Subject Target
IP	172.20.201.198
MAC	?
MAC owner	?
P0f	Linux 2.2 (1) (up: 5 hours)

Who	Gateway
IP	10.10.10.1
MAC	0:50:56:40:0:6d
MAC owner	VMWare
P0f	?

Who	Alternate Target
IP	172.20.11.2
MAC	?
MAC owner	?
P0f	?

Who	Alternate Target
IP	172.20.11.1
MAC	?
MAC owner	?
P0f	?

1.2 Detect was generated by

The 14 files are binary log files which could be generated by any number of packet capture programs like tcpdump, snort and ethereal. From what I have observed they do not seem

[68] Ian Martin's posted practical on the incidents.org mailing list – http://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00089.html
[69] Les Gordon's posted practical on the incidents.org mailing list – http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html
[70] Andre Cormier's posted practical on the incidents.org mailing list – http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00162.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

to follow the pattern as described in the file http://www.incidents.org/logs/raw/README, the `README` suggests the binary log files are made up of traffic triggered by an unknown set of snort rules. In these files almost all the packet streams are in their entirety, for example the full TCP three way hand shake to its close is intact.

The only thing missing from these capture files is any data after the 96th byte. The snap length for the capture seems to have been set to 96 bytes so a lot of the payload is missing from the trace, though there is usually enough to ascertain what is happening within a selected trace.

Whichever program captured these logs it was set to capture log files of a particular size before rotating to a new file as seen by a consistent file size of around 3Mb. The time frame for the captured data is over a 78.5 minute time span. The activity we are interested in occurs in all but two files, being the first 12 which contain approximately 25 minutes of data, the additional two files contain the remaining 50 plus minutes of data.

To analyse the binary log files I used a few common tools like tcpdump[71] (version 3.7.2), Snort[72] (version 2.1.1), Ethereal[73] (version 0.10.2) and p0f[74] (version 2.0.3). When using snort I used a default `snort.conf` with a rule file set dated 20040404, the only change being to uncomment all rule types to make them available.

1.3 Probability the source address was spoofed

Probability the source was spoofed is nearly impossible, host `10.10.10.186` actively attacks `172.20.201.198` using multiple protocols including TCP. For the TCP protocol to function correctly it needs to maintain state, maintaining state with TCP is nearly impossible to do if you spoof your IP address.

There is the remote chance that the attacker is sniffing the packets crossing the wire and spoofing every response packet, though this is not even remotely likely and would be and extremely advanced technique. The attacker does not make an effort to hide the attack as it is easily detectable, it would be a waste of effort to do this and then make the attack obvious.

1.4 Description of attack

`10.10.10.186` is seen to actively attack `172.20.201.198`, the attacker follows four distinct stages in their attempt to compromise the victim:

1. Testing of the FTP service on the target for a potential vulnerability;
2. Directed attack towards the FTP service with a buffer overflow;
3. More detailed analysis of the target host via a port scan, OS fingerprinting and vulnerability assessment; and
4. Another attack directed against the FTP service with the same buffer overflow.

The attacker uses a couple of tools in an attempt to compromise this host. Starting off with a basic FTP client to test for availability of specific commands, they then run a specific exploit with the intention to take advantage of a flaw in the FTP server to compromise the host.

The exploit is executed multiple times throughout the trace and is described in this CVE entry related to the wu-ftpd 2.6.0 vulnerability – http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0573

Further references related to this specific exploit include:

---

[71] Tcpdump homepage http://tcpdump.org/
[72] Snort home page http://www.snort.org
[73] Ethereal home page http://www.ethereal.com
[74] p0f home page http://lcamtuf.coredump.cx/p0f.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

AusCert description of a vulnerability within the site command for wu-ftpd "site exec" command for version up to and including 2.6.0 – http://www.auscert.org.au/render.html?it=1911

Cert advisory for the vulnerability.http://www.cert.org/advisories/CA-2000-13.html

The exploit code could possible be this:

wuftpd <= 2.6.0 x86/linux remote root exploit (2000/06/28) – http://www.mindsec.com/files/examples/7350wu/7350wu.c

After a few attempts using the exploit the attacker moves on to using a vulnerability scanner with the aim of discovering as much information about the target as possible, providing the attacker valuable information for further attack, though the only real exploit that comes from the attacker is the `SITE EXEC` buffer overflow.

1.5 Attack Mechanism

As previously mentioned there are four stages to this attack, here I describe each stage.

1.5.1 Stage1

In files `2003.12.15.1` to `2003.12.15.3` we find what appears at first to be some fairly benign probing of the FTP service. The attacker having connected to the FTP server probes the service for information on its ability to handle the `SITE` command. The site command is intended to be used by the administrator of the FTP server. Taken from the apache FTP server site[75]:

> <The> "SITE command is used by the server to provide services specific to the system. Most of the SITE commands can be used by the admin only. You can get all the available SITE commands by "SITE HELP".
>
> All the server administrative tasks can be performed by the SITE command. So the administrator can monitor, control the server remotely."

Here are some snippets of the payload, remember the snap length on the trace is set to 96bytes so we are unable to view the whole packet payload.

```
attack reqst =>                 SITE help
server respn =>                 214-The following SITE command
server respn =>                   UMASK          GROUP
```

In another connection we find the attacker test each `SITE` option inturn and receive an error in response, this does appear to be done by hand as seen by the timing between each request. Here I show a portion of the connection with a rounded time value, showing a 4.5 to 7 second gap between each request.

```
attack reqst =>      95.6sec    SITE index
server respn =>                 500 'SITE INDEX': command not
attack reqst =>      102.5sec   SITE minfo
server respn =>                 500 'SITE MINFO': command not
attack reqst =>      107.1sec   SITE checksum
server respn =>                 500 Nothing transferred yet
attack reqst =>      113.4sec   SITE chmod
server respn =>                 500 'SITE CHMOD': command not
```

Even though the attacker receives a negative response to the `SITE chmod` command they still try to lessen the file permission on a selected file to have full Read Write Xecute for the Owner Group and Everone, fortunately for the target this does not work.

```
attack reqst =>                 SITE chmod 777 libnss_files-2.
server respn =>                 553 Permission denied on serve
```

During this initial stage we don't find any snort rules triggered by the attacker's activity.

[75] http://incubator.apache.org/projects/ftpserver/site_cmd.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1.5.2 stage 2

After testing the FTP server for the SITE command the attacker then tries to exploit it with a buffer overflow. If we look at file `2003.12.15.4` within Ethereal at line `16801` we see a new connection to the FTP server, in this trace we see the server response that reveals the FTP server allows anonymous access and has the name "`lazy`". After the initial login we see the FTP server bombarded with a collection of strange `SITE EXEC` commands that have odd parameters, the start of this trace look like:

```
server respn =>                    220 lazy FTP server (Version w
attack reqst =>       23.02sec     USER ftp
server respn =>                    331 Guest login ok, send your
attack reqst =>       23.04sec     PASS mozilla@
server respn =>                    230 Guest login ok, access res
attack reqst =>       23.06sec     SITE EXEC %020d|%.f%.f|
server respn =>                    200-00000000000000000049|0-2|
server respn =>                    200  (end of '%020d|%.f%.f|')
attack reqst =>       23.13sec     SITE EXEC 7 mmmmnnnn%.f%.f%.f%
server respn =>                    200-7 mmmmnnnn-2-2200-20700000
server respn =>                    200  (end of '7 mmmmnnnn%.f%.f
attack reqst =>       23.27sec     SITE EXEC 7 mmmmnnnn%.f%.f%.f%
server respn =>                    200-7 mmmmnnnn-2-2200-20700000
server respn =>                    200  (end of '7 mmmmnnnn%.f%.f
```

the trace lasts for just over 27.6seconds and contains 222 packets most of which look similar to the above. With the complexity of the commands and the speed of execution it would be safe to say this trace is from an automated attack tool. The attacker seems to have kept this connection open for a while performing other tasks as a graceful close for this connection is found in a later file `2003.12.15.7`. Towards the end of this specific trace we see the attempted buffer overflow finish with this command sequence:

```
attack reqst =>       28.67sec     id;
server respn =>                    uid=0(root) gid=0(root) groups
```

This connection only triggered two snort rules

```
[**] [1:553:6] POLICY FTP anonymous login attempt [**]
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"POLICY FTP anonymous login attempt";
content:"USER"; nocase; pcre:"/^USER\s+(anonymous|ftp)/smi"; flow:to_server,established;
classtype:misc-activity; sid:553; rev:6;)
```

This rule[76] is designed to fire when a user logs into an FTP server using either the username anonymous or ftp. If this FTP server is expected to allow anonymous logins you probably would turn this rule off for traffic directed to the server otherwise you would expect to receive a lot of false positives. This rule is triggered many times throughout the attacker's activity, from now on I will treat this rule as being inactive.

```
[**] [1:1971:3] FTP SITE EXEC format string attempt [**]
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP SITE EXEC format string attempt";
flow:to_server,established; content:"SITE"; nocase; content:"EXEC"; nocase; distance:0;
pcre:"/^SITE\s+EXEC\s[^\n]*?%[^\n]*?%/smi"; classtype:bad-unknown; sid:1971; rev:3;)
```

Simply put this rule[77] is designed to trigger when the packet contains `SITE EXEC` and at least two percent (`%`) signs separated by a number of characters within the following string as long as they are not newline characters. It is designed to pick up this specific activity directed to the a WU-FTPD daemon prior to 2.6.2 and does. To understand how this command works in greater detail look at some of the references at the end of the paper. This rule correctly triggers when the buffer overflow is attempted against the FTP server and is triggered once for each attempt made by the attacker.

Looking at file `2003.12.15.4` within Ethereal at line `24339` we see the attacker makes a new connection in an attempt to verify the previously executed exploit, this connection continues into the next capture file `2003.12.15.5`. The user logs in with the user `jsmith@company.com`.

[76] Snort rule reference http://www.snort.org/snort-db/sid.html?sid=553
[77] Snort rule reference http://www.snort.org/snort-db/sid.html?sid=1971

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
server respn =>                    220 lazy FTP server (Version w
attack reqst =>       66.22sec     USER anonymous
server respn =>                    331 Guest login ok, send your
attack reqst =>       74.58sec     PASS jsmith@company.com
server respn =>                    230 Guest login ok, access res
attack reqst =>       74.58sec     SYST
server respn =>                    215 UNIX Type: L8
attack reqst =>       76.36sec     PASV
server respn =>                    227 Entering Passive Mode (172
attack reqst =>       76.40sec     LIST
server respn =>                    150 Opening ASCII mode data co
server respn =>                    226 Transfer complete.
attack reqst =>       ~122sec      QUIT
server respn =>                    221-You have transferred 0 byt
server respn =>                    221-Total traffic for this ses
```

Maybe the attacker does not see what they expect so they perform the same buffer overflow again which is in file `2003.12.15.6`. This attempt does not complete cleanly as the victim host resets the connection just before the end.

In file `2003.12.15.7` we find the final close of the first buffer overflow attempt, maybe having this connection open caused the second attempt to fail so a third attempt at the buffer overflow is made which seems to end with a desirable sequence of characters for the attacker.

```
attack reqst =>                    SITE EXEC 7 t....PsPsu....PsPs
server respn =>                    200-7 t...PsPsu...PsPsv...PsPs
attack reqst =>                    3....F3...jT...'.....=..R..h..
server respn =>                    uid=0(root) gid=0(root) groups
```

This connection is again reset by the victim host, the buffer overflow does not appear to be working for the attacker. In file `2003.12.15.8` we see the attacker lead off with a DNS lookup and then a connection to the FTP server with an abrupt but friendly close possibly just verifying the FTP server is still active on the target. I suspect the attacker is not seeing what they want so, they move onto stage 3 of the attack.

During the execution of each buffer overflow snort triggers the same `FTP SITE EXEC` rule (`SID:1971`) otherwise there are no other signs of the attackers activity.

1.5.3 Stage 3

Looking at file `2003.12.15.8` in ethereal we can see the timing of events

```
DNS Query                  4.889sec
Start of FTP connection    4.895sec
Close of FTP connection    9.373sec
DNS Query                  85.410sec
Port Scan starts           85.630sec
```

This shows when the attacker verified the FTP server was still active, then within the next 75 plus seconds' opens up another attack tool, configures it and launches it against the victim host. The attack tool starts with a DNS lookup and a full connect scan of host `172.20.201.198` which continues well into file `2003.12.15.9`. The scan begins with port 1 and scans sequentially up to port 15000.

This scan is a full connect scan which expects an open port to generate a SYN/ACK in response to the connection, using tcpdump we are able identify the ports that gave a positive response. As the port scan covers both file 8 and 9 a similar command was required to identify the open ports discovered in the rest of the scan.

```
/usr/sbin/tcpdump -nn -vv -r 2003.12.15.8 "host 10.10.10.186 and tcp[13] & 0x12 = 0x12"
```

The open ports are:

Port	Description
21	ftp
22	ssh
23	telnet
25	smtp
79	finger
98	admin package linux conf, on a Solaris box
111	sunrpc portmapper

Port	Description
1071	?
3306	mysql
6010	x11-ssh-offset 6010/tcp # SSH X11 forwarding offset
6011	?
6012	?
6013	?
6014	?

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

113	auth
513	login ?
514	command shell for tcp, udp it is known for syslog
587	submission, mail message submission
1024	?
1043	?

6015	?
6016	?
6017	?
6018	?
6020	?

During the course of this port scan additional snort rules were triggered, they are fairly basic rules that trigger when a connection is attempted to specific ports, they could generate numerous false positives depending on the wether the service is available or not, they may not be implemented in a production rule set because of the potential for a high rate of false positives[78].

```
[**] [1:1418:3] SNMP request tcp [**]
[**] [1:1420:3] SNMP trap tcp [**]
[**] [1:1421:3] SNMP AgentX/tcp request [**]
[**] [1:615:5] SCAN SOCKS Proxy attempt [**]
[**] [1:618:5] SCAN Squid Proxy attempt [**]
[**] [1:620:6] SCAN Proxy Port 8080 attempt [**]
```

Now the port scan is complete and we see what appears to be a standard nmap[79] OS finger print attempt[80], several packets with strange flag combinations hit the target system. All these packets are within log `2003.12.15.9` and can be viewed in ethereal at the line numbers shown

T1 – line `29053` – is a SYN packet with a bunch of TCP options to an open port, in our case we know that port 21 is open.

```
Source               sport      Destination          dport        Info
10.10.10.186         48599      172.20.201.198       21           [SYN, ECN] Seq=0 Ack=0
      Win=3072 Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198       21         10.10.10.186         48599        [SYN, ACK] Seq=0 Ack=1
      Win=32595 Len=0 MSS=265 TSV=1938094 TSER=1061109567 WS=0
10.10.10.186         48599      172.20.201.198       21           [TCP ZeroWindow] 48599 > 21
      [RST] Seq=1 Ack=2894031810 Win=0 Len=0
```

T2 – line `29054` – is a NULL packet with options to open port, again to port 21.

```
Source               sport      Destination          dport         Info
10.10.10.186         48600      172.20.201.198       21           [] Seq=0 Ack=0 Win=3072
      Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
10.10.10.186         48600      172.20.201.198       21           [] Seq=0 Ack=0 Win=3145728
      Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
```

T3 – line `29055` – us a SYN|FIN|URG|PSH packet with options to an open port.

```
Source               sport      Destination          dport        Info
10.10.10.186         48601      172.20.201.198       21           [FIN, SYN, PSH, URG] Seq=0
      Ack=0 Win=3072 Urg=0 Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198       21         10.10.10.186         48601        [SYN, ACK] Seq=0 Ack=1
      Win=32595 Len=0 MSS=265 TSV=1938095 TSER=1061109567 WS=0
10.10.10.186         48601      172.20.201.198       21           [TCP ZeroWindow] 48601 > 21
      [RST] Seq=1 Ack=2891563653 Win=0 Len=0
```

T4 – line `29056` – is an ACK to an open port with options

```
Source               sport      Destination          dport        Info
10.10.10.186         48602      172.20.201.198       21           [ACK] Seq=0 Ack=0 Win=3072
      Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198       21         10.10.10.186         48602        [TCP ZeroWindow] 21 > 48602
      [RST] Seq=0 Ack=1567106289 Win=0 Len=0
```

T5 – line `29057` - is a SYN packet to a closed port with options, the port scan would have revealed which ports are closed, but port 1 is a safe bet as it is almost always closed, I have never seen it legitimately open.

```
Source               sport      Destination          dport        Info
10.10.10.186         48603      172.20.201.198       1            [SYN] Seq=0 Ack=0 Win=3072
      Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198       1          10.10.10.186         48603        [TCP ZeroWindow] 1 > 48603
      [RST, ACK] Seq=0 Ack=0 Win=0 Len=0
```

T6 – line `29058` – is an ACK to a closed port with options

[78] To obtain more information on these specific rules search for the Snort ID at http://www.snort.org/cgi-bin/sigs-search.cgi
[79] nmap's man page http://www.insecure.org/nmap/data/nmap_manpage.html
[80] nmap fingerprinting techniques are described in this classic text – http://www.insecure.org/nmap/nmap-fingerprinting-article.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
Source              sport      Destination           dport        Info
10.10.10.186        48604      172.20.201.198        1            [ACK] Seq=0 Ack=0 Win=3072
      Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198      1          10.10.10.186          48604        [TCP ZeroWindow] 1 > 48604
      [RST] Seq=0 Ack=1567106289 Win=0 Len=0
```

Snort triggered on this connection with this rule because the ACK was set to zero

```
[**] [1:628:3] SCAN nmap TCP [**]

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP"; stateless; flags:A,12;
ack:0; reference:arachnids,28; classtype:attempted-recon; sid:628; rev:3;)
```

T7 – line `29059` – is a FIN|PSH|URG to a closed port with options

```
Source              sport      Destination           dport        Info
10.10.10.186        48605      172.20.201.198        1            [FIN, PSH, URG] Seq=0 Ack=0
      Win=3072 Urg=0 Len=0 WS=10 MSS=265 TSV=1061109567 TSER=0
172.20.201.198      1          10.10.10.186          48605        [TCP ZeroWindow] 1 > 48605
      [RST, ACK] Seq=0 Ack=0 Win=0 Len=0
```

Snort triggered on this connection with this rule because URG PSH and FIN bits where set.

```
[**] [1:1228:3] SCAN nmap XMAS [**]
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap XMAS"; stateless; flags:FPU,12;
reference:arachnids,30; classtype:attempted-recon; sid:1228; rev:3;)
```

PU – line `29060` – is a UDP packet to a closed port

```
Source          sport    Destination       dport    Protocol    Info
10.10.10.186    48592    172.20.201.198    1        UDP
172.20.201.198           10.10.10.186               ICMP        Destination unreachable
```

Sifting through an nmap signature file from version 3.47 I found this signature as possibly being that of the target system. To understand the makeup of this signature please refer to the Fyodor's classic fingerprinting article at http://www.insecure.org/nmap/nmap-fingerprinting-article.txt

```
Fingerprint Linux 2.3.28-33
Class Linux | Linux | 2.3.X | general purpose
TSeq(Class=RI%gcd=<8%SI=<177B202&>3C1B3)
T1(DF=Y%W=7C70%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7C70%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0|A0|0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

The attacker now has a clear idea as to which ports are open and perhaps an idea as to which TCP/IP stack is installed on the victim host. The traffic generated by the attacker is a series of tests as seen in capture files `2003.12.15.9` to `2003.12.13`. All the ports previously observed as being open by the port scan are tested for vulnerabilities. Looking at the payload for some of these tests we can determine the whole scan / fingerprinting effort is likely to have been generated by the vulnerability scanner Nessus, evidence is contained in this packet and others:

```
server respn =>                    220 lazy FTP server (Version w
attack reqst =>                    USER anonymous
server respn =>                    331 Guest login ok, send your
attack reqst =>                    PASS nessus@nessus.org
server respn =>                    230 Guest login ok, access res
server respn =>                    221 You could at least say goo
```

Due to the snap length we can't see all the information exchanged between the hosts, the ftp server name is visible, but the version has been cut off, as a guess I would expect the '`w`' to stand for wu-ftp[81] daemon from Washington University. We notice that the attacking host connects with the user anonymous and a password of `nessus@nessus.org`, a sure sign this is a connection generated by Nessus. A full nmap TCP connect() scan is part of a standard Nessus vulnerability assessment

[81] http://www.wu-ftpd.org/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Based on the banners received during the Nessus scan we can identify some of the software installed on this server which will be valuable information for the attacker.

```
Port 21 – w? (I suspect it would have indicated wu-ftp)
Port 22 – SSH-1.99-OpenSSH_2.1.1
port 23 - Red Hat Linux release 7.0
port 25 - 220 lazy ESMTP Sendmail 8.11.0
```

When port 23 is tested by Nessus it reveals the OS version, which can easily be cross checked for vulnerabilities specific to this OS. Nessus is not a stealthy scanner so it should be expected to trigger any IDS installed somewhere between the attacker and the destination, these are the types of rules triggered by the Nessus scan, many of which triggered more than once during the scan[82].

```
[**] [1:491:6] INFO FTP Bad login [**]
[**] [1:1672:7] FTP CWD ~ attempt [**]
[**] [1:1432:4] P2P GNUTella GET [**]
[**] [1:361:9] FTP SITE EXEC attempt [**]
[**] [1:489:7] INFO FTP no password [**]
[**] [1:604:5] RSERVICES rsh froot [**]
[**] [1:1992:2] FTP LIST directory traversal attempt [**]
```

1.5.4 Stage 4

Before the Nessus scan has completed the attacker takes the opportunity to run the buffer overflow exploit again. It looks very similar to the previously described `SITE EXEC` exploit, the difference with this one is what happens after the exploit has completed. The exploit seems to work, at which point the attacker begins to explore the host system. To conserve space I will only present some of the more interesting parts of the trace

In log file `2003.12.15.10` we see the full buffer overflow executed, after looking at a few directories on the server the attacker checks what user they are logged in as

```
attack reqst =>                 id
server respn =>                 uid=0(root) gid=0(root) groups
```

Then checking a documents directory we find a directory containing documents related to the version of wu-ftp installed, 2.6.0 is vulnerable to the `SITE EXEC` exploit.

```
attack reqst =>                 cd doc
attack reqst =>                 ls
server respn =>                 wu-ftpd-2.6.0
```

This trace continues in file `2003.12.15.11` where we see the attacker trying to view the `.rhosts` file, the reason for this will become apparent at the end of the trace.

```
attack reqst =>                 cat .rhosts
server respn =>                 cat:
server respn =>                 .rhosts: No such file or direc
```

This triggers a simple snort rule designed to identify if the "`.rhosts`" character sequence is seen in a packet headed towards the FTP server:

```
[**] [1:335:4] FTP .rhosts [**]
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP .rhosts"; flow:to_server,established;
content:".rhosts"; reference:arachnids,328; classtype:suspicious-filename-detect; sid:335;
rev:4;)
```

The attacker then checks many aspects of the host including network information. This command reveals all the listening services.

```
attack reqst =>                 netstat -an
server respn =>                 Active Internet connections (s
server respn =>                 0 172.20.201.198:1020     192.
server respn =>                    0      0 172.20.201.198:22
server respn =>                 ISHED tcp        0      0 172
server respn =>                                udp        0
server respn =>                 DGRAM                     563
```

the attacker is also able to view information about the makeup of the host and view users within the `/etc/passwd` file

[82] To obtain more information on these specific rules search for the Snort ID at http://www.snort.org/cgi-bin/sigs-search.cgi

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
attack reqst =>                 uname -a
server respn =>                 Linux lazy 2.2.16-22 #1 Tue Au
attack reqst =>                 cat passwd
server respn =>                 root:x:0:0:root:/root:/bin/bas
server respn =>                 che:x:48:48:Apache:/var/www:/b
```

The last part of the trace continues in file `2003.12.15.12` where the attacker changes into a new user

```
attack reqst =>                 su - jsmith
attack reqst =>                 id
server respn =>                 uid=500(jsmith) gid=100(users)
```

From here the attacker starts to investigate some of the files on the system.

```
attack reqst =>                 cd ../work*
attack reqst =>                 ls
server respn =>                 important-proposal.txt
attack reqst =>                 cat imp*
server respn =>                 Blah blah blah...\n\n(sounds imp
```

Further into the trace the attacker tries to append data to the file, I am not sure how successful the attacker is as the snap length makes it difficult to see if the data is there when the attacker reads the file back. Before disconnecting from the system the attacker tries one more command.

```
attack reqst =>                 rlogin 172.20.11.1
server respn =>                 172.20.11.1: Connection refuse
```

An attempt is made to remotely log into another system `172.20.11.1` this receives a negative response. The previously seen `.rhosts` check was related to this command, the `.rhosts` file holds the rules for external users and systems that can connect to the system using the well known "`r`" commands like `rlogin`, `rsh` etc. It is the remote systems `.rhosts` file that determines what hosts can connect.

Normally I would expect to see an attacker set up shop on the server by uploading further tools and establishing a back door for later access, instead the attacker just breaks the connection.

1.5.5 The attacker moves on?

I am not sure if the attacker succeeded in their efforts to compromise `172.20.201.198` but they move on to other targets including `172.20.11.2` and `172.20.11.1`, making a connection to port 513 (`whois` daemon) on `172.20.11.2` to find this service not listening. The attacker moves on to make a series of SSH connection on `172.20.11.1`, which is the same host they tried to use `rlogin` to open a remote connection to.

Because the attacker did not leave a back door on `172.20.201.198` when they decide to return to the host they run the `SITE EXEC` exploit again, towards the end of the trace tries to `rlogin` into `172.20.11.1` again without success. This is an attempt to relay though the victim host to another victim, the goal is to take advantage of privileges the original victim might have to manipulate the subsequent victim host.

1.6 Correlations

There are numerous hosts on the `10.10.10.x` network attacking hosts in other networks. My suspicions are this attacker is within a mock network created for users to perform attacks on various hosts, there would be great opportunity for the attacker to share information identified by other attackers or even use another host to do the reconnaissance.

The first correlations we can make are with hosts within the same `10.10.10.x` subnet the attack host is in. There are many other hosts within this subnet that target our victim like host `10.10.10.165` which performs an entire ping sweep of the IP range that `172.20.201.198` lives in. Other hosts that target `172.20.201.198` are:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
10.10.10.196, 10.10.10.234, 10.10.10.228, 10.10.10.147, 10.10.10.232, 10.10.10.142,
10.10.10.160, 10.10.10.224, 10.10.10.174, 10.10.10.214.
```

We know that Nessus is widely advertised as a security tool for the security professional to perform a vulnerability assessment of various hosts. Certain groups within the black hat community are probably using this tool as well, but because its not very stealthy I am sure the more experienced black hat is using other more difficult to detect techniques.

This link talks about another issue with wu-ftpd related to the `SITE EXEC` command but in this case is not a buffer overflow, rather a configuration issue in slackware.
http://www.wu-ftpd.org/wu-ftpd-faq.html#QA72

Wu-ftpd is not the only system to be vulnerable to `SITE EXEC` exploits:

This is a more recent SITE command vulnerability found in GlobalSCAPE Secure FTP Server (2004-03-18) http://secunia.com/advisories/11159/

While compiling the correlations section I found Chris Compton had also detailed this detect, he makes the assumption that `10.10.10.196` and `10.10.10.186` are working in tandem to penetrate the victim host. Chris's paper presents the material slightly differently providing more of a focus on the actual exploit rather than the sequence of events the attacker went through. http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00020.html

Davison Avery documented another example of this exploit found within the same log files but from a different attacking host (10.10.10.228) and victim (172.20.201.135).
http://cert.uni-stuttgart.de/archive/intrusions/2004/03/msg00097.html

1.7 Evidence of Active Targeting

The attacker is half way through an FTP connection when logging starts so there is no traffic indicating prior reconnaissance to locate this particular host. The attack seems directed to this particular host until completion of the last buffer overflow attempt.

The attacking host does not interact with any other hosts until the last couple of log files, there is again no evidence of previous reconnaissance being performed at the network layer. If I am correct in assuming this is a mock network I suspect there was wide scale sharing of reconnaissance information at the social level between all the attackers in the `10.10.10.x` network.

When we look at the source port numbers from the attacking host used with each new connection we can see they increase sequentially with little to no gaps between successive port numbers. This indicates the victim was actively targeted by the attacker and not performing any other network related tasks while targeting the victim.

1.8 Severity

(Criticality+Lethality) – (System Countermeasures+Network Countermeasures) = Severity

(4 + 5) – (0 + 0) = 9

Based on the information I have collected on these log files I suspect this is a network set up specifically to allow people to attack systems, and test their ability to compromise hosts.

Criticality = 4

A public FTP server for a corporate should have a high criticality, this service though not as visible as your Web server does provide a public face to your company.

Lethality = 5

The system is vulnerable to the exploit so a full compromise is possible via this attack vector providing a root shell.

System Countermeasures = 0

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

As highlighted by the port scan there are multiple services externally accessible, if the date on the log files is correct it was recorded late 2003, the software installed on this system is well behind in patches. I suspect this environment has been setup to allow people to test their attack techniques so system countermeasures are not implemented to make the environment a little easier to attack.

Network Countermeasures = 0

Network countermeasures again are non existent in this environment, there was no packet filtering in front of the system

1.9 Defence Recommendations

An anonymous FTP server is always open to abuse and thus should be strictly controlled and monitored. If you don't require anonymous access to your FTP server consider more secure methods of file transfer like SFTP or SCP. These provide an encrypted tunnel protecting all the data as it travels over the Internet while also providing a more secure method of authentication if using cryptographic keys.

There are still uses for anonymous FTP servers, assuming you do want to allow anonymous access to your files, FTP is an ideal candidate. If you allow anonymous access we can assume the data being offered is not high in value, administrators should make sure this is the case by removing all restricted data from the server, even if it is not being served by the FTP daemon, as a compromise of the server would still make this data available to the attacker.

FTP is a clear text protocol so all aspects of the communication can be monitored over the network, no one should be able to hide their actions if you freely allow access to the system so the protocol is probably the most ideal as you can get maximum benefit from you network IDS. In the real world I would expect this whole attack sequence to be detected by an IDS situated on the outside of the FTP server. Snort produced more than enough alarm bells for me to start an investigation.

Because of its clear text nature I would not use FTP anytime I was offering confidential data, access control should be strict and your main source of logging may need to come directly from the host itself.

The trace shows that though detectable the attacker was able to deliver what ever packet they desired to the victim without any form of filtering. Two things become apparent, there is no firewall in place, and the system has what I would consider excessive services available, especially if its home is directly on the Internet. All network services not required for the functioning of the FTP server should be removed, and a firewall implemented to block any traffic not required for public access.

System hardening resources include:

OpenNA book: "Securing & Optimizing Linux: The Hacking Solution (v3.0)" can be located at http://www.openna.com while a free earlier release can be found at: http://www.openna.com/products/books/sol/solus.php, from the website:

The SANS store http://store.sans.org/ offers papers on securing Linux like "Securing Linux: A Survival Guide for Linux Security"

Red Hat hardening script named Bastille Linux can be found at: http://www.bastille-linux.org/,

Taking the date of the trace literally (December the 15th 2003) we find the version of Linux installed on the victim is quite old, this could indicate a lack of version control, a lack of system management, or security patch policy. All these are very important in the day to day running of an Anonymous FTP server. There are many guides that can help with

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

policy creation one being "The SANS Security Policy Project" http://www.sans.org/resources/policies/ which provides a collection of policies and policy templates.

1.10 Multiple Choice Test Question

How many packets does nmap send to a host when trying to identify the OS type given the `-O` command line option:

a. 6
b. 7
c. 8
d. 9

Answer: c (T1 – T7 which are TCP tests, and a UDP test)

1.11 References

Snort User Manual which includes a section on "How to Write Snort Rules and Keep Your Sanity" http://www.snort.org/docs/snort_manual/

Detailed description of PCRE (Perl Compatible Regular Expression) as supported by snort http://www.pcre.org/pcre.txt

Mastering Regular Expressions (2nd Edition) Jeffrey E. F. Friedl (Published by O'Reilly)

1.12 Intrusions.org

There are four links related to the posted detect:

http://www.dshield.org/pipermail/intrusions/2004-May/008018.php (My original post)

http://www.dshield.org/pipermail/intrusions/2004-May/008019.php (Don Murdoch's response to my detect)

http://www.dshield.org/pipermail/intrusions/2004-May/008024.php (Responses to Don's questions)

http://www.dshield.org/pipermail/intrusions/2004-May/008026.php (Lastly Don's final comments to my response)

Here are what I consider the top three questions from Don Murdoch and my response:

```
> There is the remote chance that the attacker is sniffing the
> > packets crossing the wire and spoofing every response packet,
> > though this is not even remotely likely and would be and ..
> > extremely advanced technique.  The attacker does not make an
> > effort to hide the attack as it is easily detectable, it
> > would be a waste of effort to do this and then make the
> > attack obvious.
>
>       don - hmmm.... do you mean sniffing locally on the
>       wire or doing a man in the middle kind of thing?
```

In this case I was referring to the possibility that an attacker could be sniffing the network traffic with a sniffer anywhere on the path between the two communicating hosts, listening to the traffic as it passes performing a non blind form of the Mitnik attack. I figure as long as the address used to perform the attack is either not assigned to a real host or the real host is silenced (i.e. by a DoS) the attacker could maintain a full TCP communication with the target as long as they can see every return packet.

```
> server respn =>              220 lazy FTP server (Version w
> > attack reqst =>            USER anonymous
> > server respn =>            331 Guest login ok, send your
> > attack reqst =>            PASS nessus at nessus.org
> > server respn =>            230 Guest login ok, access res
> > server respn =>            221 You could at least say goo
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
>
>       don - ok, so you draw the conclustion that its nessus,
>       but what did you look at in the nessus app to correlate..
>       is there a particular assessment that matches the "goo"?
>       in other words, does the nessus scanner have a ftp test
>       that uses this email addr?
```

Performing a quick grep (grep -rin "nessus at nessus.org" *) on the nessus plugins for the login string "nessus at nessus.org" allows me to add this to the detect.

Nessus is a fully featured open source Vulnerability Assessment tool that provides an ever increasing set of tests to asses the security posture of a target host from over the network. Nessus uses plugins to supply information for each of these tests. One of the plugins "`logins.nasl`" does not do any security checks by itself but does provide default login configuration information for other plugins to use.

Plugin information: http://cgi.nessus.org/plugins/dump.php3?id=10870

The source for this plugin has these settings by default:
http://cvsweb.nessus.org/cgi-bin/cvsweb.cgi/~checkout~/nessus-plugins/scripts/logins.nasl?content-type=text/plain

```
default_ftp_login = "anonymous";
default_ftp_password = "nessus at nessus.org";
default_ftp_w_dir = "/incoming";
```

```
> I suspect this environment has been setup to allow people to
> > test their attack techniques so system countermeasures are
> > not implemented to make the environment a little easier to attack.
>
>       don - at this point I don't recall all of the text above,
>       but didn't you show that they were thwarted at some points?
```

I don't think I show the attacks are thwarted but more the attacks did not always complete successfully, I am not an expert in performing these types of attacks on hosts but I suspect due to the nature of buffer overflow attacks and the like they would not always work as expected 100% of the time.

Part 2.2

2 Attack host 10.10.10.142

2.1 Source of Trace

This detect was taken from files contained within the tar gzip file `2003.12.15.tgz` located at http://www.incidents.org/logs/raw/. This archive contains 14 separate binary log capture files named `2003.12.15.1` to `2003.12.15.14`.

There is a lot of activity recorded in these files from many hosts to multiple targets, I have chosen to concentrate on traffic generated by `10.10.10.142`. Traffic generated by this host is recorded within files 6 – 13 inclusive so this discussion will concentrate on these files. This host seems to target `192.168.17.135` but via another host `172.20.11.3`. Figure 17 and information on each host was constructed using techniques as identified by Ian Martin[83], who inturn used ideas courtesy of Les Gordon[84] and Andre Cormier[85].

[83] Ian Martin's posted practical on the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00089.html
[84] Les Gordon's posted practical on the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html
[85] Andre Cormier's posted practical on the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00162.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

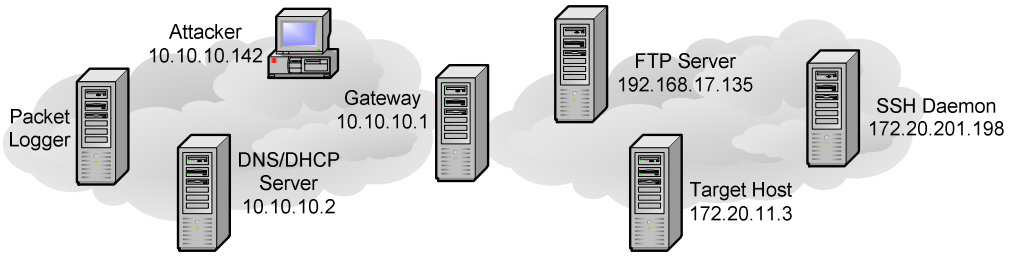


Figure 17 - Identified Network Layout

Details on the Attacking host:

Who	Attacker
IP	10.10.10.142
MAC	0:c:29:14:1e:63
MAC owner	VMWare
P0f	Linux 2.4/2.6 (up: 0 hrs)

The attack host came online some time in file `2003.12.15.6` as evidenced by this bootp (DHCP) traffic. The key packets involved in the bootp transaction are in bold with the additional packets related to retries, the first couple of attempts by the attack host to obtain an IP address where either unheard or ignored by the DHCP server

```
/usr/sbin/tcpdump -n -r 2003.12.15.6 'ether host 00:0c:29:14:1e:63 or ether host
ff:ff:ff:ff:ff:ff'

05:07:53.041037 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from
        00:0c:29:14:1e:63, length: 300
05:07:58.032847 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from
        00:0c:29:14:1e:63, length: 300
05:08:06.033149 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from
        00:0c:29:14:1e:63, length: 300
05:08:06.048346 arp who-has 10.10.10.142 tell 10.10.10.2
05:08:06.345766 IP 10.10.10.2.67 > 10.10.10.142.68: BOOTP/DHCP, Reply, length: 300
05:08:06.348248 IP 0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from
        00:0c:29:14:1e:63, length: 300
05:08:06.365744 IP 10.10.10.2.67 > 10.10.10.142.68: BOOTP/DHCP, Reply, length: 300
05:08:07.044113 arp who-has 10.10.10.142 tell 10.10.10.2
05:08:07.044840 arp reply 10.10.10.142 is-at 00:0c:29:14:1e:63
05:08:07.045315 IP 10.10.10.2 > 10.10.10.142: icmp 28: echo request seq 0
05:08:07.045765 IP 10.10.10.142 > 10.10.10.2: icmp 28: echo reply seq 0
```

From this trace, we can see the networks DHCP server is `10.10.10.2`. The attack host is an OS hosted within a VMWare[86] session that was identified by looking up the registered[87] owner of the MAC address. The OS type was identified by a p0f[88] scan across the traffic generated by the attacking host, and the OS type has been identified as being a version of Red Hat Linux[89] by some regularly occurring DNS requests, an example looks like:

Source	sport	Destination	dport	Info
10.10.10.142	32770	10.10.10.2	53	Standard query A www.rhns.redhat.com
10.10.10.142	32770	10.10.10.2	53	Standard query A www.rhns.redhat.com.attackers.org

Not only does this show that `10.10.10.2` also provides DNS services for the network but the attack host is trying to resolve an IP address for Red Hat's "Red Hat Network"[90] service offered by Red Hat to keep users system up to date. A default installation of Red Hat installs a small program named up2date[91] which is kept running while the system is active.

`10.10.10.142` attempts a connection to RHN every 60 seconds, these queries are not being satisfied leading me to suspect this network does not have an Internet connection providing an upstream DNS server to satisfy the request. Because the first query does not resolve, a second attempt uses the same URL with an appended search path

[86] VMWare home page http://www.vmware.com/
[87] Search who has registered the MAC address http://standards.ieee.org/regauth/oui/index.shtml
[88] More information of the P0f passive scanner can be located at http://freshmeat.net/projects/p0f/
[89] Red Hat Linux home page http://www.redhat.com/
[90] Red Hat's Red Hat network service https://rhn.redhat.com/
[91] Up2date download site - https://rhn.redhat.com/help/latest-up2date.pxt

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

`attackers.org`. This search parameter was probably added to the `resolve.conf`[92] file during the DHCP process to ease DNS lookups from the user perspective. Other players in the trace include:

Who	DNS/DHCP server
IP	10.10.10.2
MAC	0:50:56:40:0:64
MAC owner	VMWare
P0f	?

Who	Target Host
IP	172.20.11.3
MAC	?
MAC owner	?
P0f	?

Who	Gateway
IP	10.10.10.1
MAC	0:50:56:40:0:6d
MAC owner	VMWare
P0f	?

Who	SSH Daemon
IP	172.20.201.198
MAC	?
MAC owner	?
P0f	?

Who	FTP Daemon
IP	192.168.17.135
MAC	?
MAC owner	?
P0f	Linux 2.4/2.6 (up: 13 hrs)

2.2 Detect was generated by

In all three detects I use the same source of data as described in detect one under section 1.2, the 14 files are binary log files which could be generated by any number of packet capture programs. These files only have the first 96 bytes causing a lot of the payload to be missing from the selected trace. Each log file was rotated to the next when it reached around 3Mb, the total period captured by these is around 78.5 minutes.

Tools I used to analyse the binary log files included tcpdump[93] (version 3.7.2), Snort[94] (version 2.1.1) and Ethereal[95] (version 0.10.2). When using snort I used a default `snort.conf` with rule file set dated 20040404, the only change being to uncomment all rule types to make them available to the snort rule processor.

2.3 Probability the source address was spoofed

The attacker having the IP address `10.10.10.142` uses `192.168.17.135` to proxy a port scan through this host. If the tool the attacker used to perform the scan was a little stealthier and the proxy host allowed the attacker to perform the requested action, the victim host `172.20.11.3` would never have seen packets from the attackers IP address yet the attacker would have been able to glean some very valuable information about the target host.

From the perspective of the `192.168.17.135` host, it is unlikely the attacker's source address is being spoofed as the main connection to this host is FTP which uses TCP as the transport protocol. For the TCP protocol to function correctly it needs to maintain state, maintaining state with TCP is nearly impossible to do if you spoof your IP address.

The attacker attempts to hide their identity behind the `192.168.17.135` address when trying to scan `172.20.11.3`, with adequate logging on the FTP server this connection would be logged allowing the target system administrators to contact the owner of the FTP server and potentially correlate events to track down the attacker's ultimate IP address.

There is a remote chance the attacker is sniffing the packets as they cross the wire headed towards a real host, and spoofing all the response packets. This would require the attacker to spoof an unused IP address or perform a DoS on the legitimate owner so it cannot interrupt the connection. This is not even remotely likely and would be and extremely advanced technique.

[92] Detailed explanation of the resolve.conf file http://www.rt.com/man/resolver.5.html
[93] Tcpdump homepage http://tcpdump.org/
[94] Snort home page http://www.snort.org
[95] Ethereal home page http://www.ethereal.com

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.4 Description of attack

There are two different modes available to FTP, “Active mode” and “Passive mode”. The described attack commonly known as “FTP bounce” relates to the way FTP manages active mode FTP connections. A great explanation on the two FTP modes is located at http://slacksite.com/other/ftp.html.

In short FTP requires two network connections to function, one is the command channel (port 21 on the server) the other is the data channel (in active FTP this is port 20 on the server). The command channel is required in both modes and opened by the client, how the data channel is established is where the two modes differ. In passive mode, the FTP client is also responsible for establishing the data channel with the FTP server. FTP servers that use active mode FTP require the client to specify an IP address and port number for which the FTP server will actively establish the data channel back to the client.

Here lays the problem, in some cases the client can specify arbitrary IP addresses and ports for the FTP server to connect too, these don’t always have to be associated with the connecting client in any way. This allows the attacker to proxy network scans through the FTP server thus hiding their true source.

If this attack worked, an attacker could hide their activities behind the FTP servers IP address making the attacker extremely difficult to locate. Port scanning, bypassing packet-filtering devices and bypassing export restrictions are just some of the possibilities. If the FTP server also allows a writable directory then you have even more potential. A file containing SMTP commands could be uploaded to the FTP server, the attacker would use the PORT command to connect to a mail server they could then upload the file to the mail server causing it to relay email, providing only the FTP servers IP address as the source.

2.5 Attack Mechanism

After obtaining an IP address, the attacker connects to a SSH server on `172.20.201.198` and maintains this connection while on the network. My thoughts are this network is constructed for people to test attack and penetration techniques and this host could contain instructions or information on the network, many other hosts make lengthy connections to this server all made without prior reconnaissance.

Because there are no significant snort rules triggered by the attack host, I identified the traffic manually by searching the trace with ethereal. The attack appears automated as it is repeated 3 times using the same sequence and as we will see in a very short time span, Figure 18 shows the connections made by the attacker, each one will be described shortly. The specific trace we are looking at is in file `2003.12.15.12`, opening this file in ethereal allows us to obtain line numbers and packet timing.

Attacker host is `10.10.10.142`
Target host is `172.20.11.3`
Proxy host is `192.168.17.135`

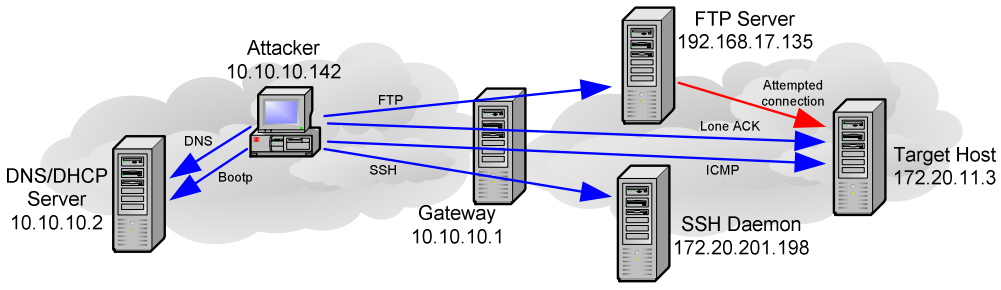


Figure 18 - Connections made by the attacker

Whatever software was used to generate the trace it begins with a ping to the target host.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

line	Time	src ip	src prt	dst ip	dst prt	protocol	Comments
20302	155.833sec	10.10.10.142	-	172.20.11.3	-	icmp	echo request
20306	155.850sec	172.20.11.3	-	10.10.10.142	-	icmp	echo reply

At almost the same time, the attacker sends a lone ACK packet destined for port 80 typically known as the HTTP port, the source port is 53 well-known for DNS. If part of a full TCP connection, this traffic would still be suspicious due to the port combinations.

line	Time	src ip	src prt	dst ip	dst prt	protocol	Comments
20303	155.833sec	10.10.10.142	53	172.20.11.3	80	tcp	lone ack packet
20307	155.850sec	172.20.11.3	80	10.10.10.142	53	tcp	rst

Source Port 53 is possible but unusual, more commonly seen in older implementations of DNS, normally an ephemeral port is used as the source port. Additionally TCP is not normally used by DNS, but there are two circumstances where it is:

1. When the requesting host requires information that exceeds the capacity set for a UDP connection of 512bytes; or
2. During a Zone transfer.

As there is no other related traffic from the attacker, I suspect this is an attempt to bypass a stateless packet filter that could be sitting in front of the target host. The target responds to the attacker when receiving the ICMP echo request and the lone ACK packet. The ICMP echo request elicits a desirable response as far as the attacker is concerned, an echo reply shows the host is alive. The lone ACK packet generates a RST, which is an expected result for two reasons:

1. The HTTP service is not listening on port 80; or
2. The host does not have any knowledge on a previous connection related to this TCP packet, when initiating a connection a lone ACK does not start the negotiation process, it is seen in the last part of a three way handshake (SYN, SYN/ACK, ACK). As there is no previous state information the target just resets the connection.

The attacker now knows the target host is available, a reverse DNS lookup of the target IP address to obtain the FQDN (Fully Qualified Domain Name) of the host via the local DNS server `10.10.10.2`. Due to the 96-byte snap length, we cannot see the full DNS response to identify the name the target IP address would resolve as.

line	Time	src ip	src prt	dst ip	dst prt	protocol	Comments
20312	156.200sec	10.10.10.142	32770	10.10.10.2	53	dns	ptr request 172.20.11.3
20313	156.203sec	10.10.10.2	53	10.10.10.142	32770	dns	ptr response

The focus of this detect is when the attacker connects to the FTP service on a very different host `192.168.17.135`. Here is the TCP three way hand shake:

line	Time	src ip	src prt	dst ip	dst prt	protocol	Comments
20314	156.255sec	10.10.10.142	39471	192.168.17.135	21	tcp	syn
20317	156.532sec	192.168.17.135	21	10.10.10.142	39471	tcp	syn,ack
20318	156.533sec	10.10.10.142	39471	192.168.17.135	21	tcp	ack

Once the connection is established the attacker logs into the FTP server using anonymous as seen by the username and password combination supplied to the server. Using the FTP `PORT` command the attacker attempts to open a connection to the target host.

```
attack reqst =>      163.546sec       USER anonymous
server respn =>                       220 suse72all.target.labs.veri
attack reqst =>      166.589sec       PASS -wwwuser@
server respn =>                       331 Guest login ok, type your
server respn =>                       230 Guest login ok, access res
attack reqst =>      168.624sec       PORT 172,20,11,3,0,144
server respn =>                       500 Illegal PORT rejected (res
attack reqst =>      168.918sec       PORT 172,20,11,3,13,129
server respn =>                       500 Illegal PORT rejected (add
server respn =>                       221 You could at least say goo
```

A RST by the FTP server closes the connection. We can see how quickly the attack was by the addition of timing information.

line	Time	src ip	src prt	dst ip	dst prt	protocol	Comments
22294	168.962sec	192.168.17.135	39471	10.10.10.142	21	tcp	rst

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The port command is used to open a data channel with the current Active FTP server. As mentioned in 2.4 - Description of attack there are two different modes an FTP server can use, Active mode or Passive mode.

In our example, the attacker is trying to take advantage of an issue associated with servers that implement Active FTP. When a client connects to an active FTP server they make the standard initial connection to port 21, when the client requires the transfer of data it issues a "`PORT`" command, this tells the FTP server which port to connect back on. Take note, it is the FTP server that makes the initial connection from port 20 back to the client, the client just informs the FTP server which destination port to connect to the client on.

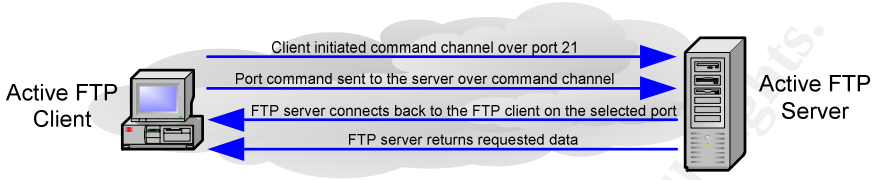


Figure 19 - Basic Flow of an Active FTP Connection

The Client has the responsibility to tell the FTP server, which IP address and port it is required to connect back to the client on, potentially the client could ask the FTP server to make a connection back to any accessible host. In our case, the attacker tries to make the FTP server to establish a data connection back to the target host using this command.

```
PORT 172,20,11,3,0,144
```

What does this command mean? The six decimal numbers separated by comers make up the IP address and port number for the return connection, the obvious part is the first four decimal numbers that are the same as the target IP address `172.20.11.3`. In a legitimate connection, the client opens up this port awaiting a return connection from the FTP server, the last two decimal numbers are the destination port. To calculate the port we take the left number and multiplying it by 256 and then add the right number.

i.e. `(0 x 256) + 144 = 144`

As another example, we can take the next PORT command

```
PORT 172,20,11,3,13,129
```

This has the same destination IP address of the target but a destination port number being `(13 x 256) + 129 = 3457`

In both attempts, the FTP server rejects the attempt:

```
500 Illegal PORT rejected (add
```

Even though the command does not work, there are two further attempts to perform this attack, with exactly the same attack sequence, ping, lone ACK, PTR lookup, and then FTP connection. If we consider the initial ping as time zero the speed of the scan is obvious.

```
Echo Request        0.000sec
Lone ACK            0.001sec
DNS PTR lookup      0.368sec
FTP connection      0.423sec
```

I suspect this attack was delivered by an automated tool like nmap[96] that has an FTP bounce[97] switch for this function, though I did not observe similar traffic when I tested it with my version of nmap (3.47). Unless specifically told otherwise nmap commonly pings the target host before scanning to ensure it is available before wasting resources on a down system, the initial ping matches this signature. Additionally, unless the ports are specified on the command line, nmap will scan for well-known ports as detailed in its `nmap-`

[96] nmap's man page http://www.insecure.org/nmap/data/nmap_manpage.html
[97] Description of nmap's FTP bounce switch http://www.insecure.org/nmap/nmap_doc.html#bounce

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

`services` file, ports 141 and 3457 are listed in this file, ports are usually scanned randomly in an effort to evade detection hence the reason they are out of sequence.

2.6 Correlations

This attack is commonly know as an FTP bounce, the attack itself is very old and well known, the possible uses of the FTP bounce attack, and vulnerable FTP daemons are well documented. In July 1995 *Hobbit* wrote a paper on the FTP Bounce Attack, this is the original source describing the possibilities and abuses that can be had with an FTP server, this paper is well worth the read. http://www.insecure.org/nmap/hobbit.ftpbounce.txt

The reason this has had such a large impact in the past is related to the RFC standards[98]. This behaviour is compliant with the RFC standard, which allows clients to choose the IP address and Port number thus opening up any RFC compliant FTP daemon to this vulnerability, technically, it would be following the standard.

A later RFC (RFC2577[99]) describes the security implications of the original standards[100] and suggests ways to mitigate the impact of the attack, further discussion in the CERT Coordination Centre paper titled "Problems with the FTP PORT Command or Why You Don't Want Just Any PORT in a Storm" initially released in April 1998.

http://www.cert.org/tech_tips/ftp_port_attacks.html

The Common Vulnerabilities and exposure database have a couple of entires relating to the FTP bounce vulnerability, the most important being this generic CVE entry.

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0017

This link lists numerous vendor products that are vulnerable to the exploit, a specific example is the FTP proxy in Symantec Raptor Firewall 6.5.3 and Enterprise 7.0 – http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0538

This page lists systems that are vulnerable to the Symantec Raptor / Enterprise Firewall FTP Bounce issue described by CVE 2002-0538 (last updated Apr 17, 2002) http://www.securityfocus.com/bid/4522

More information on multiple Vendor FTP Bounce Attack Vulnerability (last updated Jul 10, 2003) http://www.securityfocus.com/bid/126

There are advisories related to numerous vulnerable systems, most of which are quite old with the most recent issues in 2002, showing this vulnerability is still relevant even seven years after the *Hobbit* article. Examples of the most recent exposures include:

IRIX ftpd Bounce vulnerability, Thursday Mar 28, 2002 http://www.securitywarnings.com/warnings/?id=20

CA's Eserv FTP server denial of service and bounce attack vulnerabilities (discovered January 29, 2002) – http://www3.ca.com/threatinfo/vulninfo/Vuln.aspx?ID=4800

2.7 Evidence of Active Targeting

Based on all the traffic generated by the attack host, both the victim and the proxy host were targeted. The FTP proxy host does not see any traffic from the attacker except this small attack pattern directed to the FTP server except to port 21 in any of the other files indicating prior reconnaissance.

```
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ; do /usr/sbin/tcpdump -n -r 2003.12.15.$i 'dst
host 10.10.10.142 and src host 192.168.17.135 and tcp[13] & 0x12 == 0x12' ; done | awk -F "
" '{print $3}' | awk -F "." '{print $5}' | sort | uniq | sort -n

21
```

[98] A conmprehensive list of the FTP related RFC standards http://www.networksorcery.com/enp/default0601.htm
[99] FTP Security Considerations (May 1999) http://www.faqs.org/rfcs/rfc2577.html
[100] http://www.networksorcery.com/enp/default0602.htm details all the relevant FTP RFC's

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

A different host could have identified the FTP server, or information sharing took place. I suspect this attacker is within a mock network constructed for people to exercise their penetration testing skills. In such an environment, there would be great opportunity to share information identified by other attackers or use another host for reconnaissance.

If the attacker did indeed use a tool to generate this traffic, they had prior knowledge of the parameters for the tool before running it. There are numerous hosts on the `10.10.10.x` network attacking hosts within other networks many of which connect to `172.20.201.198` for long periods, these connections do not appear to be attacks, this host could hold information guiding attackers on what is available within the mock network.

2.8 Severity

(Criticality+Lethality) – (System Countermeasures+Network Countermeasures) = Severity

(4 + 3) – (2 + 1) = 4

Ultimately, the vulnerability is with the FTP server so this is related to the severity of allowing the PORT command to be used as a proxy to run commands on other systems.

Criticality = 4

A corporate public FTP server should have a high criticality, this service is usually not as visible as your Web Server but does provide a public face for your company.

Lethality = 3

There are quite a few mischievous things an attacker can do if they are able to find an FTP server that allows an FTP bounce to occur as described in the *Hobbit* article. The attacks though not necessarily damaging to the FTP server itself pose a serious threat to the credibility of the company hosting the server, as they will appear to be the source of any mischievous activity the attacker performs.

System Countermeasures = 2

It is hard to determine if there are any serious system countermeasures on the FTP server. Even though the FTP server allows anonymous access the fact the FTP server is not susceptible to this attack reduces the severity of the attack and increases the system countermeasures. I suspect that in this specific environment there are no countermeasures to make the environment a little easier to attack.

Network Countermeasures = 1

There does not appear to be any Network based countermeasures protecting this host. The fact snort did not trigger an alert with a default rule set probably increases the severity and the fact the ftp server allows anonymous access probably increases the alertness and vigilance required by the owners of this server.

2.9 Defence Recommendations

Within documents cited in this paper there are a few recommendations including restricting the use of the PORT command, or if still required only allowing the PORT command to connect to ports greater than 1024. An obvious solution is not to run a vulnerable FTP daemon in the first place.

A vulnerability scanner should vigorously test any host on your network before being placed into production. This includes any third party products like printers that have an FTP daemon, or any system that proxies FTP traffic like a proxy server or an application layer Firewall.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Many of the Security scanners on the market will test for the FTP bounce vulnerability like ISS Security Scanner[101], Retina[102], SARA[103] and Nessus[104]. Nessus is the only totally free[105] security scanner mentioned and has a test for the FTP bounce vulnerability with script id 10081, described at http://cgi.nessus.org/plugins/dump.php3?id=10081

2.10 Multiple Choice Test Question

During an active FTP connection the client sends a "`PORT`" command to the FTP server requesting the establishment of a data channel, what IP address and port number does the client give the FTP server to establish this return connection?

```
PORT 172,20,11,3,23,162
```

a. IP: `11.3.23.162` port: `44052`
b. IP: `172.20.11.3` port: `6050`
c. IP: `11.3.23.162` port: `17220`
d. IP: `172.3.11.3` port: `23162`

Answer: b (The process of converting this number is described in section 2.5)

2.11 References

Most references are embedded within the document

Example trace files of legitimate traffic, this can be used for comparison and provide understanding the way traffic patterns should appear. http://www.packet-level.com/traceFiles.htm

Part 2.3

3 Attack host 10.10.10.165

3.1 Source of Trace

This detect was taken from files contained within the tar gzip file `2003.12.15.tgz` located at http://www.incidents.org/logs/raw/. This archive contains 14 separate binary log capture files named `2003.12.15.1` to `2003.12.15.14`.

There is a lot of activity recorded in these files, I have chosen to concentrate on traffic generated by the subject host `10.10.10.165`. Traffic generated by this host is recorded within files 1 – 13 but the specific packets in this discussion are in file `2003.12.15.5`. Figure 20 and information on each host was constructed using techniques as identified by Ian Martin[106], who inturn used ideas courtesy of Les Gordon[107] and Andre Cormier[108].

[101] ISS Security Scanner can be found through ISS main web site http://www.iss.net/
[102] Retina Security Scanner can be found at http://www.eeye.com/html/Products/Retina/index.html
[103] SARA Security Scanner can be found at http://www-arc.com/sara/
[104] Nessus home page http://www.nessus.org
[105] The free software definition http://www.gnu.org/philosophy/free-sw.html
[106] Ian Martin's posted practical on the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2003/07/msg00089.html
[107] Les Gordon's posted practical on the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00221.html
[108] Andre Cormier's posted practical to the incidents.org mailing list - http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00162.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

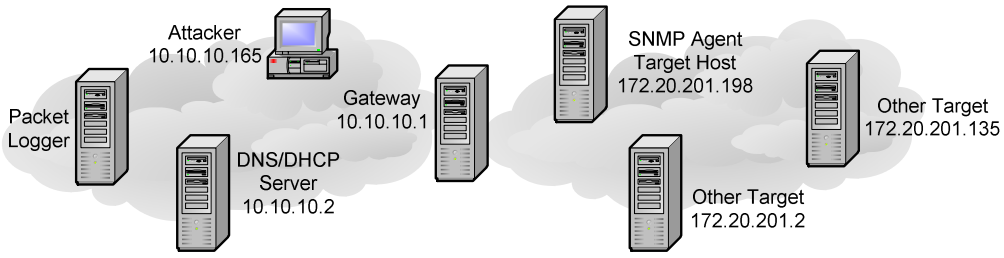


Figure 20 - Identified Network Layout

Details on the Attacking host and target:

Who	Attacker
IP	10.10.10.165
MAC	0:3:47:8c:89:c2
MAC owner	Intel Corporation
P0f	Windows 2000 SP2+, XP SP1 (seldom 98 4.10.2222), Windows XP/2000 while downloading (leak!)

Who	SNMP Agent (target)
IP	172.20.201.198
Telnet banner	Red.Hat.Linux.release.7.0.(Guinness)
P0f	?

Command used to generate trace

```
/usr/sbin/tcpdump -nn -r 2003.12.15.5 'host 10.10.10.165 and port 161'
```

For hex dumps I added the "`-X`" command line parameter to the above command. Selected examples of packets between `10.10.10.165` and `172.20.201.198` are:

```
05:07:00.578222 IP 10.10.10.165.1672 > 172.20.201.198.161:  C=FirstBogus
       GetRequest(26)  .1.3.6.1.2.1.1.1.0

05:07:04.587904 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetRequest(26)  .1.3.6.1.2.1.1.1.0
05:07:04.632204 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.1.0="Linux"

05:07:04.758182 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetNextRequest(26)  .1.3.6.1.2.1.1.1.0
05:07:04.864461 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.2.0=.1.3.6.1.4.1
05:07:04.864965 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetNextRequest(26)  .1.3.6.1.2.1.1.2.0
05:07:04.935582 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(33)  .1.3.6.1.2.1.1.3.0=1904931
05:07:04.936019 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetNextRequest(26)  .1.3.6.1.2.1.1.3.0
05:07:05.070605 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
05:07:05.071095 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetNextRequest(26)  .1.3.6.1.2.1.1.4.0
05:07:05.137609 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(34)  .1.3.6.1.2.1.1.5.0="lazy"
05:07:05.138005 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetNextRequest(26)  .1.3.6.1.2.1.1.5.0
05:07:05.234561 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.6.0="Unkno"

05:07:10.986227 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetRequest(26)  .1.3.6.1.2.1.1.4.0
05:07:10.989554 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
05:07:11.491741 IP 10.10.10.165.1672 > 172.20.201.198.161:
       SetRequest(39)  .1.3.6.1.2.1.1.4.0="iss.net Root "
05:07:11.685548 IP 172.20.201.198.161 > 10.10.10.165.1672:  GetResponse(35)
       noSuchName@1 .1.3.6.1.2.1.1.4.0="iss.n"
05:07:13.186188 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetRequest(26)  .1.3.6.1.2.1.1.4.0
05:07:13.196139 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
05:07:13.196470 IP 10.10.10.165.1672 > 172.20.201.198.161:  SetRequest(39)  .1.3.6.1.2.1.1.4.0="Root
       <root@lo"
05:07:13.203665 IP 172.20.201.198.161 > 10.10.10.165.1672:  GetResponse(35)
       noSuchName@1 .1.3.6.1.2.1.1.4.0="Root "
05:07:14.703008 IP 10.10.10.165.1672 > 172.20.201.198.161:  GetRequest(26)  .1.3.6.1.2.1.1.4.0
05:07:14.706116 IP 172.20.201.198.161 > 10.10.10.165.1672:
       GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
```

Here we see host `10.10.10.165` connecting to `172.20.201.198` on UDP port 161 (SNMP). The target host has a listening SNMP service accepting connections with a community string of "`public`". The full trace shows the attacker "walking" the SNMP tree to discover as much information as possible about the host, potentially revealing information about vulnerabilities the host may have and giving the attacker more information than desired.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.2 Detect was generated by

In all three detects I used the same source of data as described in detect one under section 1.2, the 14 files are binary log files which could be generated by any number of packet capture programs. These files only have the first 96 bytes causing a lot of the payload to be missing from the selected trace. Each log file was rotated to the next when it reached around 3Mb, the total period captured by these is around 78.5 minutes

Tools I used to analyse the binary log files included tcpdump[109] (version 3.8.3), Snort[110] (version 2.1.2) and Ethereal[111] (version 0.10.3). When using snort I used a default `snort.conf` with rule file set dated 20040529, the only change being to uncomment all rule types to make them available to the snort rule processor.

3.3 Probability the source address was spoofed

In previous detects I suggested this network was constructed for people to test their penetration skills, with that in mind this traffic is very unlikely to be spoofed. However, to be a little more scientific, the network protocol used to target our victim host is UDP:

```
05:07:00.578222 IP 10.10.10.165.1672 > 172.20.201.198.161:  C=FirstBogus
GetRequest(26)  .1.3.6.1.2.1.1.1.0
        0x0000:  4500 0049 8d46 0000 8011 22d4 0a0a 0aa5  E..I.F....".....
        0x0010:  ac14 c9c6 0688 00a1 0035 67fa 302b 0201  .........5g.0+..
        0x0020:  0004 0a46 6972 7374 426f 6775 73a0 1a02  ...FirstBogus...
        0x0030:  0256 8402 0100 0201 0030 0e30 0c06 082b  .V.......0.0...+
        0x0040:  0601 0201 0101 0005 00
```

UDP is a connectionless protocol that does not have the mechanisms to maintain state information on connections. If state is required it relies on the upper level protocols (e.g. within the application) to maintain this information. This gives UDP the ability to be very simple and efficient in its delivery of packets but leaves network security devices in a state of ambiguity as to whether each successive connection between the same hosts and UDP ports is a continuation from a previous packet or an entirely new connection. This makes it very hard at a network level to determine if the packet was spoofed.

To help we need to try to understand the attackers motivation, some possibilities include:

1. is the attacker trying to perform a denial of service against the host;
2. does the attacker think it if fun to just send random packets and not expect a response; or
3. is it used to discover information about the target

In scenario 1 an attacker would be wise to hide their identity, though in this trace there are not enough packets to put any sort of strain on the system, making it is unlikely to be a resource exhaustion form of DoS attack. There have been documented cases where a single packet can disable a system[112], these DoS attacks use packets that are malformed in some way and these appear correct removing that possibility.

Only a very bored individual could find scenario 2 interesting leaving the most likely scenario, “the attacker wants more information about the target”. To obtain the required information you need responses, to see the response you need a real IP address.

There is a remote chance the attacker is sniffing packets as they cross the wire headed towards a real host, and spoofing the response packets. An attacker would need to spoof an unused IP address or perform a DoS on the legitimate owner so it cannot interrupt the connection. This is very unlikely and would be an extremely advanced technique.

[109] Tcpdump homepage http://tcpdump.org/
[110] Snort home page http://www.snort.org
[111] Ethereal home page http://www.ethereal.com
[112] The ping of death – http://www.insecure.org/sploits/ping-o-death.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.4 Description of attack

This is not necessarily an attack per-se but an attempt at gathering intelligence on the targeted system, normally a prelude to an attack. After obtaining enough information during the reconnaissance phase, the attacker would specifically construct an attack based on information. According to "the Hack FAQ"[113] there are four basic steps to hacking a system, the abridged steps are[114]:

1. Learn as much as possible about your target before the attack;
2. Initial access to the system;
3. Obtain full system access; and lastly
4. Cover your tracks and install backdoors.

This trace shows an example of step one, the attacker is attempting to learn as much as they can about the target system before launching into step 2 were they will attempt to achieve initial system access.

3.5 Attack Mechanism

Only looking at the traffic generated between the subject hosts within file `2003.12.15.5` we can identify what SNMP alerts snort triggered on and the frequency.

```
/usr/bin/snort -c ./rules/snort.conf -de -r 2003.12.15.5 -l ./172.20.201.198/ 'host
172.20.201.198 and host 10.10.10.165'
grep '\[\*\*\]' alert | sort | uniq -c | sort -r | head
     40 [**] [1:1411:3] SNMP public access udp [**]
      4 [**] [1:1417:2] SNMP request udp [**]
```

The first packet is shown under section 3.3 along with a subsequent attempt that looked very similar there was no response from the target host, these packets trigger snort rule 1417[115] which fires on packets directed to port 161. The next packet shows another attempt to communicate with the target host using the community string "public":

```
05:07:04.587904 IP 10.10.10.165.1672 > 172.20.201.198.161:
GetRequest(26)  .1.3.6.1.2.1.1.1.0
        0x0000:  4500 0045 8e74 0000 8011 21aa 0a0a 0aa5  E..E.t....!.....
        0x0010:  ac14 c9c6 0688 00a1 0031 1fdc 3027 0201  .........1..0'..
        0x0020:  0004 0670 7562 6c69 63a0 1a02 0256 8502  ...public....V..
        0x0030:  0100 0201 0030 0e30 0c06 082b 0601 0201  .....0.0...+....
        0x0040:  0101 0005 00                             .....
```

This is the default community string used by the SNMP protocol and is the trigger for snort rule 1411[116], the target system obviously uses the same community string as we receive a positive response to the connection, which also triggers snort rule 1411.

```
05:07:04.632204 IP 172.20.201.198.161 > 10.10.10.165.1672:
GetResponse(35)  .1.3.6.1.2.1.1.1.0="Linux"
        0x0000:  4500 0086 942c 0000 3e11 5db1 ac14 c9c6  E....,..>.].....
        0x0010:  0a0a 0aa5 00a1 0688 0072 e32d 3082 0066  .........r.-0..f
        0x0020:  0201 0004 0670 7562 6c69 63a2 8200 5702  .....public...W.
        0x0030:  0256 8502 0100 0201 0030 8200 4930 8200  .V.......0..I0..
        0x0040:  4506 082b 0601 0201 0101 0004 394c 696e  E..+........9Lin
        0x0050:  7578                                     ux
```

Remember the snap length for this trace was set to 96 bytes so we are missing the remaining component to this trace. There is enough here for us see the attacker just identified the OS version of the SNMP agent is running on (`Linux`).

The long number "`.1.3.6.1.2.1.1.1.0`" is called an OID (Object ID)[117] and is a series of numbers that uniquely identify a location of information within an SNMP agent. The attacker is looking at the system OID, which contains information regarding the target

[113] http://www.nmrc.org/pub/faq/hackfaq/index.html
[114] http://www.nmrc.org/pub/faq/hackfaq/hackfaq-02.html
[115] http://www.snort.org/snort-db/sid.html?sid=1417
[116] http://www.snort.org/snort-db/sid.html?sid=1411
[117] http://www.adventnet.com/products/snmputilities/help/quick_tour/snmp_and_mib/snmpmib_miboverview.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

system, you always want to know what system you are targeting before you attack, so this would make sense as a first step.

The requested OID can be broken down like this (`[.1.3.6.1.] [2.1.1.] [1.] [0]`):

OID element	Description	notes
.1.3.6.1.	MIB II	1 = Iso 3 = org 6 = dod 1 = internet
2.1.1.[118]	System	
1.	System MIB elements	1 – sysDescr This number can be changed to look at other items *2 – sysObjectID* *3 – sysUpTime* *4 – sysContact* *5 – sysName* *6 – sysLocation* *7 - sysServices*
0	first instance of this OID	

The attacker then continues to look for further information on the target host by "walking" the SNMP tree, this is where you continually request the next available information item. The attacker's software sends an SNMP GETNEXT request to obtain the value within the next OID in the tree, with each subsequent GETNEXT request, the attacker is walking the SNMP tree.

```
attack reqst =>    GetNextRequest(19)  .0.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.1.0="Linux"      sysDescr OID
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.1.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.2.0=.1.3.6.1.4.1 sysObjectID OID
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.2.0
server respn =>    GetResponse(33)  .1.3.6.1.2.1.1.3.0=1904931      sysUpTime OID
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.3.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "      sysContact IOD
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.4.0
server respn =>    GetResponse(34)  .1.3.6.1.2.1.1.5.0="lazy"       sysName OID
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.5.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.6.0="Unkno"      sysLocation OID
attack reqst =>    GetNextRequest(26)  .1.3.6.1.2.1.1.6.0
server respn =>    GetResponse(31)  .1.3.6.1.2.1.1.8.0=0
```

This continues we get to what seems to be the end of this sequence of OID's

```
attack reqst =>    GetNextRequest(28)  .1.3.6.1.2.1.1.9.1.4.9
server respn =>    GetResponse(32)  noSuchName@1 .1.3.6.1.2.1.1.9.1.4.9=
```

Then the attacker uses a different tactic, instead of just enumerating as much information about the host as possible, they starts to test the host. They test their ability to write to the SNMP Agent using an SNMP PUT command and then check the results by reading back the targeted OID.

```
attack reqst =>    GetRequest(26)  .1.3.6.1.2.1.1.4.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
attack reqst =>    SetRequest(39)  .1.3.6.1.2.1.1.4.0="iss.net Root "
server respn =>    GetResponse(35)  noSuchName@1 .1.3.6.1.2.1.1.4.0="iss.n"
attack reqst =>    GetRequest(26)  .1.3.6.1.2.1.1.4.0
server respn =>    GetResponse(35)  .1.3.6.1.2.1.1.4.0="Root "
attack reqst =>    SetRequest(39)  .1.3.6.1.2.1.1.4.0="Root <root@lo"
server respn =>    GetResponse(35)  noSuchName@1 .1.3.6.1.2.1.1.4.0="Root "
```

This does not seem to work, as the response is not what the attacker attempted to write. Something else to note with the last part of the trace, we can identify that the tool used in the attack as ISS security scanner[119], the attacker is scanning multiple hosts at the same time with this tool. I do not have access to the tool to verify how it functions but it does leave its mark in a similar way throughout the trace when checking other services. In file `2003.12.15.1`, we see evidence of the very first step in the attack, this is an ICMP echo request that shows more evidence ISS security scanner was used.

[118] http://www.et.put.poznan.pl/snmp/mib2/mgroup20.html
[119] http://www.iss.net/products_services/enterprise_protection/vulnerability_assessment/scanner_internet.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
/usr/sbin/tcpdump -nn -X -r 2003.12.15.1 'host 10.10.10.165 and host 172.20.201.198 and
icmp'
04:59:08.077424 IP 10.10.10.165 > 172.20.201.198: icmp 24: echo request seq 8960
        0x0000:  4500 002c 0bc0 0000 8001 a487 0a0a 0aa5  E..,............
        0x0010:  ac14 c9c6 0800 3b30 87db 2300 ac14 c9c6  ......;0..#.....
        0x0020:  3bdc 2300 4953 5350 4e47 5251 0000       ;.#.ISSPNGRQ..
```

These packets trigger a default Snort rule 465[120] that looks for this specific sequence of character in an ICMP packet "ISSPNGRQ", the detailed information for this rule suggests:

> An echo request that originates from a host running Internet Security Scanner "pinger" software contains a unique payload in the message request

3.6 Correlations

SNMP is has proven to be a very valuable tool for the network administrator and is commonly used to centrally mange devices within a network and provides a source of statistics that help in understanding the health of your network[121]. There are many well-identified issues with the most commonly used versions on SNMP (version 1 and 2). One of the drawbacks is they provide very little in the way of real security, they are clear text protocols and provide security via a single character string as the password throughout the enterprise known as a community name.

We can see the ease of identifying the community name within documented traces, but it is very common for people not to even change the community name from the well-known default of "public" making sniffing it off the network unnecessary. Products that ship with SNMP turned on sometimes use alternate community names but these are also generally well known.

On the SANS top 20 list of vulnerabilities, SNMP is mentioned as an issues in both the Windows and UNIX category. These links provide a detailed description on how to tackle problems associated with SNMP, how to determine if you are vulnerable and how to protect yourself from SNMP issues.

Windows description of the problem – http://www.sans.org/top20/#w10

UNIX description of the problem – http://www.sans.org/top20/#u7

For an even more comprehensive description of SNMP issues, how it works, its vulnerabilities, and how to protect against these attacks is provided by SANS.

http://www.sans.org/resources/idfaq/snmp.php

Guofei Jiang titles an alternate reference document that discusses the insecurities in SNMP "Multiple Vulnerabilities in SNMP". <Review some of these documents>

http://www.computer.org/security/supplement1/jia/

As part of their "PROTOS - Security Testing of Protocol Implementations" project the University of Oulu developed a SNMPv1 protocol test suite. This project discovered many issues in the implementation of SNMP finding it was not very secure.

> The initial results from the c06-snmpv1 tests indicate that implementation errors plague several SNMP products. None from the sample of twelve implementations survived the test-material. This is most alarming since SNMPv1 is widely used in critical parts of network infrastructure.

PROTOS Test-Suite: c06-snmpv1 – http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/

CVE and CAN entries related to SNMP

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0013

---

[120] ICMP ISS Pinger – http://www.snort.org/snort-db/sid.html?sid=465

[121] http://compnetworking.about.com/library/glossary/bldef-snmp.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
CAN-2002-0013 (under review) - Vulnerabilities in the SNMPv1
request handling of a large number of SNMP implementations allow
remote attackers to cause a denial of service or gain privileges
via (1) GetRequest, (2) GetNextRequest, and (3) SetRequest
messages, as demonstrated by the PROTOS c06-SNMPv1 test suite.
```

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0017

```
CVE-2002-0017 - Buffer overflow in SNMP daemon (snmpd) on SGI
IRIX 6.5 through 6.5.15m allows remote attackers to execute
arbitrary code via an SNMP request.
```

Windows – CVE-1999-0294, CVE-1999-0815, CAN-1999-0499, CAN-2002-0053

UNIX – CVE-2001-0236, CVE-2002-0797 CAN-1999-0186, CAN-1999-0254, CAN-1999-0516, CAN-1999-0517, CAN-1999-0615, CAN-2002-0012, CAN-2002-0796

3.7 Evidence of Active Targeting

As previously mentioned ISS Security Scanner generated this trace, it appears the whole network range of `172.20.201.x` was added to the scanner, which performed a vulnerability assessment on all the hosts across the specified C class network. This indicates the attacker did not specifically target the victim. In file `2003.12.15.1` we see a ping sweep from the attacker targeting this range with four hosts responding:

```
/usr/sbin/tcpdump -n -r 2003.12.15.1 'host 10.10.10.165 and icmp[icmptype] = icmp-echoreply'
04:59:08.312519 IP 172.20.201.2 > 10.10.10.165: icmp 24: echo reply seq 8960
04:59:08.312559 IP 172.20.201.198 > 10.10.10.165: icmp 24: echo reply seq 8960
04:59:08.321034 IP 172.20.201.135 > 10.10.10.165: icmp 24: echo reply seq 8960
04:59:08.420027 IP 172.20.201.1 > 10.10.10.165: icmp 24: echo reply seq 8960
```

These IP addresses once identified as alive are targeted with a raft of attacks.

3.8 Severity

(Criticality+Lethality) – (System Countermeasures+Network Countermeasures) = Severity

(3 + 2) – (2 + 1) = 2

Criticality = 3

Allowing SNMP through to Internet facing systems is a very bad idea, especially with well-known community strings like the target host. The fact that the target host does not allow the attacker to adjust the content of the SNMP OID's reduces the criticality of this attack, it does however leave the host open to detailed identification, providing enough information for the attacker to formulate a potentially successful attack.

Lethality = 2

The attack by itself if not lethal to the system, rather the ramifications of allowing this type of access to an outsider could be potentially lethal.

System Countermeasures = 2

There does not appear to be any real counter measures on the system, the system however does not allow write access to the SNMP OID's which could be considered a system countermeasure.

Network Countermeasures = 1

There does not appear to be any Network based countermeasures protecting this host as the scanner identifies many open ports, many of which should not be exposed on a protected host situated directly on the Internet.

```
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ; do /usr/sbin/tcpdump -n -r 2003.12.15.$i 'dst
host 10.10.10.165 and src host 172.20.201.198 and tcp[13] & 0x12 == 0x12' ; done | awk -F "
" '{print $3}' | awk -F "." '{print $5}' | sort | uniq | sort -n
21,22,23,25,79,98,111,113,513,514,587,1024,3306,36130
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.9 Defence Recommendations

Standard vulnerability assessment tools like ISS Security Scanner[122], Retina[123], SARA[124] and Nessus[125] will test for vulnerabilities in your SNMP implementation, this trace is actually ISS Security Scanner performing such tests. There are also many tools designed to test SNMP deployments specifically, especially since the discovery that SNMP was vulnerable at the design level. Solarwinds has a tool that checks the security of your SNMP deployment: http://www.solarwinds.net/Tools/Security/Printer_SNMP_Security.htm

This link also discusses issues related to securing SNMP and how to select a sensible community string for network environment:

- Do NOT use the default "public" or "private"
- Do NOT use something that would be easy to guess (your company name, phone number, etc..)
- Do NOT use a text only string (make it alphanumeric)
- DO use an alphanumeric string (one that contains both numbers and letters)
- DO use both upper and lower case (community strings are case sensitive)
- DO use a community string that is at least six characters in length

http://probing.csx.cam.ac.uk/about/snmp.html shows some of the steps required to disable some of the most common SNMP agents, it also mentions the default SNMP community strings used by these products.

The SANS references cited in section 3.6 also detail the process of defending yourself from SNMP issues, these usually break down to a few key principles:

- Do not allow access to SNMP from the internet, if possible filter access to this port from all systems except the manager even internally to your network.
- Use a strong community string that is unique to your network
- Ensure end systems only responds to messages from known hosts
- Send Authentication Trap - When a device receives an authentication that fails, a trap is sent to a management station.[126]
- Ensure end systems only respond to the most recent protocol available
- If at all possible use SNMPv3

3.10 Multiple Choice Test Question

What is the most common SNMP community name used to read information from an SNMP agent?

a. secure
b. private
c. public
d. read

Answer: C

Most implementations use "public" as a standard community string for read-only access.[127]

[122] ISS Security Scanner can be found through ISS main web site http://www.iss.net/
[123] Retina Security Scanner can be found at http://www.eeye.com/html/Products/Retina/index.html
[124] SARA Security Scanner can be found at http://www-arc.com/sara/
[125] Nessus home page http://www.nessus.org
[126] http://www.comptechdoc.org/independent/networking/guide/netsnmp.html
[127] http://probing.csx.cam.ac.uk/about/snmp.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Analyse This

1 Executive Summary

UMBC commissioned a security audit of their network to assist them in understanding the security posture of the UMBC network. The audit was performed through the analysis of five consecutive days of snort capture files recorded using an older snort system running an unknown rule set. There where three file types, Alert, Scan and OOS (out of spec) files, these where manipulated to generate reports used to identify unusual network patterns some of which are discussed within this analysis.

The alerts observed in these log files do provide a reasonable view into the health of the UMBC network, though they only show traffic entering or leaving the University network. With no internal network traffic available for analysis it is hard to determine if there are any inter network scanning or compromise of host with worms or Trojans.

However, only giving an overview as a full analysis would require more time than is feasible, this document does detail results from this analysis. There are quite a few false positive recorded, when identified as such the analysis provides a few recommendations that may reduce these and improve the value of future assessments given a similar time frame to perform the assessment. There are also hosts with very suspicious activity, these hosts have been detailed for University staff to verify and rectify as appropriate.

Overall, the university network is reasonably clean considering the openness and size of the environment covering a full class B network totalling 65,536 addresses. Table 9 shows that only 84 of the possible 256 /24 networks sent and or received alerted traffic during the selected period, reducing the address space to a possible 21,504 addresses. As expected with this size of network and the obvious openness due to the nature of Universities there are signs of nefarious activity.

Spoofed packets where observed indicating the University does not implement ingress or egress filtering[128] which is highly recommended. There does appear to be a serious worm propagating through the University network "W32.Mockbot.A.Worm", hosts suspected of infection is detailed within the analysis. The evidence of worm and Trojan activity indicate some users are not as security conscious as they could be, and some systems are not up to date with patching, recommendations for educating users are at the end of the analysis.

1.1 Suspicious internal hosts

Throughout this document, suspicious internal and external hosts are highlighted for further action, Table 8 shows only 10 of the identified hosts.

Table 8 - Suspicious Internal Hosts

Host IP	FQDN (Fully Qualified Domain Name)	Suspicions
130.85.150.44	illiad.lib.umbc.edu.	Opaserv or BugBear worms
130.85.69.254	lib-69-254.pooled.umbc.edu.	P2P client
130.85.17.45	erk177pc-1.umbc.edu.	W32.Mockbot.A.Worm
130.85.80.224	pplant-80-224.pooled.umbc.edu.	W32.Mockbot.A.Worm
130.85.153.195	SOA (UMBC3.umbc.edu.)	W32.Mockbot.A.Worm
130.85.111.51	trc208pc-02.engr.umbc.edu.	W32.Mockbot.A.Worm
130.85.112.193	SOA (UMBC3.UMBC.EDU.)	W32.Mockbot.A.Worm
130.85.80.119	ss-80-119.pooled.umbc.edu.	Kibuv.b Worm
130.85.97.12	ppp-012.dialup.umbc.edu.	Kibuv.b Worm
130.85.112.193	SOA (UMBC3.UMBC.EDU.)	Kibuv.b Worm

[128] http://www.hackinglinuxexposed.com/articles/20030213.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1.2 Log Files Analysed

The source of the files analysed was from http://www.incidents.com/logs

Alert logs	start time	end time
alert.040420	04/20-12:50:51	04/21-00:06:02
alert.040421	04/21-00:16:03	04/22-00:07:21
alert.040422	04/22-00:00:01	04/23-00:07:21
alert.040423	04/23-00:16:04	04/24-00:07:19
alert.040426*	04/26-00:30:43	04/26-00:15:25

Scans logs	start time	end time
scans.040420	Apr 20 13:00:31	Apr 20 23:52:25
scans.040421	Apr 21 00:00:01	Apr 21 23:57:20
scans.040422	Apr 22 00:00:01	Apr 22 23:55:15
scans.040423	Apr 23 00:00:02	Apr 23 23:56:48
scans.040426*	Apr 26 00:08:42	Apr 26 00:08:55

OSS logs	start time	end time
oos_report_040420	04/24-00:05:25	04/24-05:56:12
oos_report_040421	04/25-00:05:42	04/25-05:47:20
oos_report_040422	04/26-00:05:48	04/26-05:51:37
oos_report_040425	04/29-00:05:32	04/29-05:57:01
oos_report_040426	04/30-00:05:02	04/30-05:57:21

* These files where unusually small and corrupted, what data I could extract from them was quite short. There is limited data of use during the month of April.

1.3 Defensive Recommendations

There was a lot of data generated over this five-day period requiring analysis, this level of information is well beyond any single person to analyse on a day to day basis, even a small team of people would find it hard to keep up. To minimise the workload involved in the investigation process, the network should be broken down into segments (or critical hosts). Within large corporate environments, network segments and or hosts are categorised by risk or importance. The university could perform a risk assessment on each segment to determine which are more critical for operations, from there the level of scrutiny imposed on a are more critical segment would increase while spending less time on less critical systems.

As a University, there is a fine line between implementing more secure systems and restricting users' freedoms. It is advised to segment the network at a physical layer to separating networks that require more freedom with those that require more protection. As an example, the network segment `130.85.97.x` appears to provide access to users who have dialled into the University network and should have open access to the Internet, while the `130.85.1.x` network holds some critical infrastructure (i.e. DNS, SMTP servers etc) these should have considered protection and controls. If they are not already, it is advised these networks are separated to provide finer grained security.

As this analysis was commissioned to look at the data set as a whole with no specific information regarding sensitivity of each network segment, the following discussions focus on the data set as a whole. Take note, all scan data was removed from the alert logs (as per the process performed by Ian Martin) before generating this and other tables based on the alert data as it is repeated in the scan specific logs.

Table 9 - Alerts by Network Segment

IP	Alerts src	Alerts dst	Scans src	Scans dst	OOS src	OOS dst
1	**-**	**771**	**4702868**	**14670**	**-**	**-**
2	-	45	-	14526	-	-
4	-	21	-	14037	-	-
5	125	2243	-	13846	-	68
6	5	595	-	16126	-	983
7	-	14	-	14224	-	-
9	-	450	-	1862	-	-
10	4	184	-	14080	-	-
11	5723	110	43	2842	-	-
12	86	719	79	16418	6	868
13	-	31	-	13893	-	-
14	-	25	-	13560	-	11
15	1	644	-	14418	-	-
16	-	6	-	4576	-	-
17	14	3476	1179209	13590	-	3

IP	Alerts src	Alerts dst	Scans src	Scans dst	OOS src	OOS dst
67	-	61	-	1098	-	-
69	3010	1972	589329	7189	-	3
70	716	3522	16379	14820	-	10
71	123	218	444	13984	-	-
73	-	9	-	6653	-	-
75	528	446	1776	13719	-	-
80	89	346	989123	13392	-	-
81	1	741	694881	6762	-	-
82	196	1495	8742	7197	-	-
83	-	1290	41094	6777	-	1
84	36	567	464366	6750	-	-
86	-	10	-	6670	-	-
97	**84**	**5763**	**414290**	**58633**	**-**	**93**
98	8	61	34772	14014	-	-
99	-	224	-	13382	-	11

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

18	-	269	-	15377	-	-
20	-	25	-	13674	-	-
21	-	25	-	13452	-	-
22	-	74	-	1864	-	-
24	268	2103	-	15812	-	206
25	409	465	82347	5290	-	69
27	2	1622	92	51455	-	-
28	-	1	-	2462	-	-
29	106	1064	-	12013	-	-
30	-	51401	21	5585	-	-
31	-	1212	-	11563	-	-
32	-	6553	-	10109	-	-
33	-	-	-	1615	-	-
34	111	190	154722	2221	-	85
40	-	131	5	508	-	-
41	-	26	-	13656	-	-
42	-	22	-	13650	-	-
43	6232	6025	838857	14267	-	90
53	30	2474	151978	14143	-	-
54	-	19	-	13668	-	1
55	-	23	172	13476	-	-
56	-	13	-	3374	-	-
60	22	371	55	13433	-	18
62	5	130	-	1178	-	-
64	-	23	-	1567	-	-
65	-	347	-	3462	-	-
66	83	37	11366	8506	-	-

100	-	38	-	13099	-	-
101	-	32	-	12968	-	-
102	-	23	-	13229	-	-
103	-	12	-	3244	-	-
109	94	306	69119	13227	-	-
110	-	625	89183	13860	-	-
111	196	2385	583100	13561	-	-
112	36	911	1269298	15465	-	-
120	-	12	-	12449	-	-
121	-	33	-	12524	-	-
123	-	-	-	318	-	-
130	-	26	-	13160	-	-
136	-	6	-	382	-	-
147	-	101	-	6971	-	-
149	-	12	13	13692	-	-
150	1076	1363	227398	13573	-	4
151	8	399	16	13230	-	-
152	49	10	110	14054	-	-
153	963	1809	226717	14942	-	-
156	-	17	-	13465	-	-
165	-	16	-	13489	-	-
166	-	-	-	842	-	-
185	-	25	-	12648	-	-
186	-	14	-	13238	-	-
189	4	220	-	3386	-	12
190	178	334	-	38784	-	-
191	-	125	-	13211	-	-

1.4 Description of the analysis process

Demonstration of the analysis process used to reach the conclusions is a requirement of the paper. The layout is very similar to that used by Pete Storm[129], he had a very clean and readable paper, but unlike Pete, I do not have the database skills to import and manipulate the alert data within a database. I instead populated my tables using UNIX commands such as `sed`, `awk`, `grep`, `wc`, `cat`, originally based on the work by Ian Martin[130] in his paper.

I know this process was not ideal, it is error prone and long winded but served my purposes based on the skill set I have. In the end, these steps where probably quicker for me than setting up a database, learning how to import the data so it is usable, and learn a new language to query the database, as I don't have any prior knowledge in this area. The big draw back was the lack of flexibility to produce detailed conclusions about the data and better correlations between hosts due to the limitation of this technique.

This section describes how I prepared the data files before the analysis. The steps to create the table in sections 2.1, 2.3 and 2.5 are in Appendix A – The process, some of the additional steps used to arrive at my conclusions are included throughout the paper. Prior to performing any log analysis I needed to clean the log files, this was a long manual process using a few UNIX commands to assist.

1.4.1 Alert Logs

There are many comments in previous practicals relating to the quality of these logs, they regularly have overlapping or truncated entries so I parsed each file with `grep`, looking for lines that did not start with the date "`04/2`"

```
grep -v -n ^04\/2 alert.040420
```

This provided the line number of each entry that failed the test, from here I manually edited each file fixing the identified issue. I could not see an easy way to automate this process, the issues were with alerts spliced across multiple lines, there could be four, five or a couple of hundred lines apart, the distance was random.

[129] http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf
[130] http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

As an example, we have a situation like:

```
04/20-14:09:05.820172  [**] EXPLOIT x86 NOOP [**] 24.84.58.6304/20-14:37:14.073435  [**]
       spp_portscan: portscan status from MY.NET.1.4: 11 connections across 11 hosts: TCP(0),
       UDP(11) [**]
:4095 -> MY.NET.70.74:80
04/20-14:37:14.081775  [**] spp_portscan: portscan status from MY.NET.17.45: 98 connections
       across 21 hosts: TCP(98), UDP(0) [**]
```

It should have looked like:

```
04/20-14:09:05.820172  [**] EXPLOIT x86 NOOP [**] 24.84.58.63:4095 -> MY.NET.70.74:80
04/20-14:37:14.073435  [**] spp_portscan: portscan status from MY.NET.1.4: 11 connections
       across 11 hosts: TCP(0), UDP(11) [**]
04/20-14:37:14.081775  [**] spp_portscan: portscan status from MY.NET.17.45: 98 connections
       across 21 hosts: TCP(98), UDP(0) [**]
```

This could indicate there are multiple processes writing to the same file, each process also has a different idea of the time or there is a delay in processing before the files are recorded in the file. Possibly we have multiple processes on separate systems with unsynchronised clocks writing to the same file. In this example, we can see the time stamp is not consistent, this sample could indicate three processes writing to the same file? Andre Cormier came to a similar conclusion[131].

```
04/22-02:46:28.156289  [**] spp_portscan: End of portscan from MY.NET.81.39: TOTAL time(7s)
       hosts(58) TCP(61) UDP(0) [**]
04/22-02:30:14.905962  [**] SMB Name Wildcard [**] MY.NET.11.4:1971 -> 64.12.24.35:65535
04/22-02:14:31.293257  [**] High port 65535 tcp - possible Red Worm - traffic [**]
       MY.NET.43.8:137 -> 210.120.128.117:137
04/22-02:46:28.701969  [**] spp_portscan: PORTSCAN DETECTED from MY.NET.81.39 (THRESHOLD 12
       connections exceeded in 0 seconds) [**]
04/22-02:14:31.299999  [**] High port 65535 tcp - possible Red Worm - traffic [**]
       64.12.24.35:65535 -> MY.NET.43.8:1971
```

The cleaned files where saved with a `.clean` extension, and placed into one single file.

```
cat alert.040420.clean alert.040421.clean alert.040422.clean alert.040423.clean
   alert.040426.clean >> alert.all
```

The network these sensors are on is a class B network, with the alert logs sanitised to protect the identity of the network owners, strangely the scan logs have not. In previous practicals, the identity of the University was revealed as the University of Maryland Baltimore County[132] and the MY.NET address range as `130.85.0.0/16`. Each analyst took a different path, some choosing to maintain MY.NET in the alert data[133], some changed it with a fictitious IP address like 10.10[134] and some as I did chose to replace MY.NET with the correct IP address[135]:

```
sed 's/MY.NET/130.85/g' alert.all >> alert.all.clean
```

Then following the process Ian Martin used I separated the port scan alerts from the rest:

```
grep "spp_portscan" alert.all.clean >> alert.spp
grep -v "spp_portscan" alert.all.clean >> alert.misc
```

To ease manipulation I converted the "`[**]`" into "`:`" as a field separator

```
sed 's/ *\[\*\*\] /:/g' alert.misc  >> alert.misc1
sed 's/ *\[\*\*\] /:/g' alert.spp >> alert.spp1
```

Then for the miscellaneous alerts, I converted the "`->`" into a "`:`".

```
sed 's/ \-> /:/g' alert.misc1  >> alert.misc2
```

To show the evolution of each entry, the original format looked like:

```
date  time           alert name                 source IP+port         dest IP+port
04/20-12:50:51.940732 [**] SMB Name Wildcard [**] 130.85.150.198:1109 -> 67.162.149.143:137
```

Adjusted format indicating the individual fields:

```
date  time           alert name        source IP+port      dest IP+port
04/20-12:50:51.940732:SMB Name Wildcard:130.85.150.198:1109:67.162.149.143:137
{1}     {2}{3}       {4}               {5}            {6}  {7}            {8}
```

[131] http://www.giac.org/practical/GCIA/Andre_Cormier_GCIA.pdf
[132] http://www.giac.org/practical/GCIA/Andrew_Jones_GCIA.pdf
[133] Pete Storm
[134] http://www.giac.org/practical/GCIA/Donald_Parker_GCIA.pdf
[135] Ian Martin

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

This worked for most of the alerts in the alert file, I originally assumed each alert entry included ports, however this was not the case. There were fragmentation and ICMP alerts that do not have port entries, or a place holder. For manipulation, each entry needed to be of the same format. The first step was to separate the two types of alerts, those with port numbers and those without. Firstly, I remove all the entries that do not have port numbers:

```
cat alert.misc2 | grep '[0-9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+\.[0-9]\+\.[0-9]\+\.[0-
   9]\+$' > alert.misc.ip
```

In addition, placed the rest into a separate file:

```
cat alert.misc2 | grep -v '[0-9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+\.[0-9]\+\.[0-9]\+\.[0-
   9]\+$' > alert.misc.transport
```

There were still corrupt lines in the `alert.misc.transport` file, I knew what I expected each entry to look like `date:description:src-ip:src-port:dst-ip:dst:port` so I created a filter to match all conforming lines placing all but corrupt lines into a new file, this removed an additional 9 entries:

```
grep '04\/2[0-6]\-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+:[^\:]\+:[0-9]\+\.[0-9]\+.[0-9]\+\.[0-
   9]\+:[0-9]\+:[0-9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+' alert.misc.transport >
   alert.misc.transport2
```

Using a similar filter, the entries in `alert.misc.ip` were verified for non-corrupt entries.

```
grep '04\/2[0-6]\-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+:[^\:]\+:[0-9]\+\.[0-9]\+.[0-9]\+\.[0-
   9]\+:[0-9]\+:[0-9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+' alert.misc.ip | head
```

`alert.misc.ip` is manipulated to give it the same format as `alert.misc.transport`, the null port entries where replaced with a "`none`" comment as a place holder.

```
awk -F: '{print $1":"$2":"$3":"$4":"$5":none:"$6":none"}' alert.misc.ip > alert.misc.ip2
```

Showing the original format:

```
date and time         alert name                      source IP        destination IP
04/20-12:42:38.217125:ICMP SRC and DST outside network:172.200.101.156:58.128.231.103
{1}     {2}{3}       {4}                             {5}              {6}
```

new format looked like:

```
date and time         alert name                      source IP+port       dest IP+port
04/20-12:42:38.217125:ICMP SRC and DST outside network:172.200.101.156:none:58.128.231.103:none
{1}     {2}{3}       {4}                             {5}              {6}  {7}            {8}
```

I then joined both files again totalling 130,303 alert entries:

```
cat alert.misc.transport2 alert.misc.ip2 > alert.misc3
```

Now the alert data is ready for processing.

1.4.2 Scan Logs

The scan logs went through a similar process as the alert data, in short here are the steps:

1. Attempted to clean up the log files

   ```
   grep -v -n ^Apr\  scans.040423
   ```

2. Created one file that was in excess of 910Mb in size containing 13,816,238 individual entries

   ```
   cat scans.040420.clean scans.040421.clean scans.040422.clean scans.040423.clean scans.040426
      >> scans.all
   ```

3. First changed the "`->`" into a "`:`".

   ```
   sed 's/ \-> /:/g' scans.all >> scans.clean
   ```

4. Added a "`:`" between the destination port and the scan type information to aid my ability to manipulate the data

   ```
   cat scans.clean | sed 's/\(:[0-9]*\) /\1:/g' > scans.clean2
   ```

5. Verified all scan entries conformed to expectations, removing 3 more lines.

   ```
   grep '^Apr 2[0-6] [0-9]\+:[0-9]\+:[0-9]\+:[0-9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+:[0-
      9]\+\.[0-9]\+.[0-9]\+\.[0-9]\+:[0-9]\+:[^\:]\+' scans.clean2 > scans.clean.csv
   ```

The scan data evolved as such, the original format looked like:

```
date   time     src IP+port           dst IP+port       type of scan
Apr 20 13:00:31 130.85.34.14:34798 -> 198.247.172.10:25 SYN ******S*
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

With the adjusted format indicating the fields that can be filtered on:

```
date   time     src IP+port        dst IP+port       type of scan
Apr 20 13:00:31:130.85.34.14:34798:198.247.172.10:25:SYN ******S*
{1}      {2}{3}{4}          {5}   {6}            {7}{8}
```

Now the scan data is ready for processing.

1.4.3 OOS Logs

The OOS log files posed a bigger problem for me as they contained multi line entries making standard UNIX tools unable to cope if looking for information that crossed multiple lines. Luckily the OOS files where much smaller than the Alert and scan files, only containing 2,542 entries, small enough to manually clean out obviously corrupt entries, and create a single file from the five days of data named `oos_report.all.clean`. To create the tables in section 2.5, modification of each entry is done from the command line and piped through simular commands as the other log types.

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | [more commands]
```

An example entry looks like:

```
04/24-00:05:25.200503 204.92.130.31:39436 -> 130.85.12.6:25
TCP TTL:48 TOS:0x0 ID:37508 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x9CF95DC4  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 259922879 0 NOP WS: 0
```

With the end result looking like

```
04/24-00:05:25.200503:204.92.130.31:39436:130.85.12.6:25
{1}     {2}{3}       {4}          {5}   {6}        {7}
```

Now the OOS data is ready for processing.

2 Detailed Analysis

There are three data types requiring detailed analysis, alert, scan and OOS (out of spec) data. Each type are discussed over the next six sections, the first section presents the data in tables displaying the top offenders within selected categories, the following section then looks at selected events in greater detail. The detailed discussion will provide information on suspicious hosts that may require further investigation from University staff to verify identified concerns.

2.1 Alert Summary

The tables in this section display the alert data in various ways helping identify suspicious activity, some of which is further discussed in section 2.2

2.1.1 Alerts by Type

Fifty-Six different types of alerts where triggered during the selected period logging 130,303 times. Table 10 shows the frequency of each alert type and the approximate percentage of the total alert count.

Table 10 - Alerts by Type

Message	No.	%
EXPLOIT x86 NOOP	38953	30
130.85.30.4 activity	35321	27
High port 65535 tcp - possible Red Worm - traffic	19494	15
130.85.30.3 activity	15911	12
SMB Name Wildcard	8638	7
Tiny Fragments - Possible Hostile Activity	4420	3
RFB - Possible WinVNC - 010708-1	2360	2
Null scan!	1938	1
NMAP TCP ping!	869	<1
Possible trojan server activity	419	<1
SUNRPC highport access!	254	<1
connect to 515 from outside	250	<1

Messages	No.	%
Attempted Sun RPC high port access	25	<1
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	23	<1
External RPC call	23	<1
[UMBC NIDS] External MiMail alert	17	<1
TFTP - External TCP connection to internal tftp server	17	<1
FTP DoS ftpd globbing	15	<1
EXPLOIT NTPDX buffer overflow	14	<1
DDOS mstream client to handler	14	<1
[UMBC NIDS] Internal MiMail alert	11	<1
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	10	<1
IRC evil - running XDCC	7	<1
TFTP - Internal TCP connection to external tftp server	6	<1

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

TCP SMTP Source Port traffic	191	<1	EXPLOIT x86 stealth noop	5	<1
DDOS shaft client to handler	147	<1	connect to 515 from inside	4	<1
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	138	<1	DDOS mstream handler to client	4	<1
Incomplete Packet Fragments Discarded	127	<1	Traffic from port 53 to port 123	3	<1
High port 65535 udp - possible Red Worm - traffic	122	<1	SYN-FIN scan!	3	<1
TCP SRC and DST outside network	80	<1	Probable NMAP fingerprint attempt	2	<1
SMB C access	75	<1	NIMDA - Attempt to execute root from campus host	2	<1
FTP passwd attempt	69	<1	NETBIOS NT NULL session	2	<1<1
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	65	<1	HelpDesk 130.85.70.49 to External FTP	2	<1
ICMP SRC and DST outside network	43	<1	External FTP to HelpDesk 130.85.70.49	2	<1
TFTP - Internal UDP connection to external tftp server	41	<1	EXPLOIT x86 NOPS	2	<1
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	38	<1	[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	1	<1
[UMBC NIDS IRC Alert] Possible drone command detected	34	<1	TFTP - External UDP connection to internal tftp server	1	<1
NIMDA - Attempt to execute cmd from campus host	34	<1	External FTP to HelpDesk 130.85.70.50	1	<1
EXPLOIT x86 setuid 0	28	<1	External FTP to HelpDesk 130.85.53.29	1	<1
EXPLOIT x86 setgid 0	26	<1	Back Orifice	1	<1

2.1.2 Top 20 Alert Destination Ports

Here are the top 20 destination ports used out of 1,583 that triggered alerts.

Table 11 - Top 20 Alert Destination Ports

Port	Count	%	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
80	40786	31	World Wide Web HTTP	11	2293	824	13	42
51443	28792	22	?	1	10	1	0	0
524	14946	11	NCP	2	23	2	0	0
65535	10093	8	[trojan] Adore worm, [trojan] RC1 trojan, [trojan] Sins	3	38	27	28	60
137	8638	7	NETBIOS Name Service	2	0	1	62	639
none	4463	3	?	2	26	18	0	37
8009	3786	3	Unassigned, Novell Netware Remote Manager	2	2	2	0	0
1971	2259	2	NetOp School	1	1	1	0	0
1605	2018	2	Salutation Manager (Salutation Protocol)	1	1	1	0	0
0	1749	1	Reserved, many vulnerabilities	3	101	58	0	0
2894	1521	1	abacus-remote	6	7	2	0	0
3019	1393	1	Resource Manager	1	2	1	0	0
1759	1354	1	SPSS License Manager	2	1	1	1	1
5900	1160	<1	Virtual Network Computer	3	2	12	3	2
2718	1049	<1	pn-requester2	2	1	1	1	1
25	808	<1	Simple Mail Transfer	9	40	5	17	53
53	560	<1	Domain Name Server	3	86	5	0	0
32771	279	<1	FileNET RMI, Sometimes an RPC port on my Solaris box (rusersd)	2	31	46	0	0
515	254	<1	Spooler, [trojan] lpdw0rm, [trojan] Ramen	2	1	1	1	1
27374	227	<1	[trojan] SubSeven, [trojan] Lion, and others	1	1	1	11	33

2.1.3 Alert Reflexive Ports and IP Addresses

With a few exceptions, it is uncommon to see source and destination ports the same, it is certainly an issue when the source and destination IP addresses are the same. There where no reflexive IP addresses in the Alert data, but there where reflexive ports all detailed in Table 12, these unusual packets are potentially worth further investigation.

Table 12 - Alert Reflexive Port Combinations

Port	Count	Known use	Comments	Src IP count	Dst IP count
137	7396	NETBIOS Name Service	This is quite common traffic	61	412
0	1738	Reserved, many vulnerabilities	Port 0 is unusual at the best of times	98	55
53	208	Domain Name Server	There are some legitimate cases for this	5	2
25	191	Simple Mail Transfer	This is not normal	4	1
80	142	World Wide Web HTTP	This is not normal	83	35
65535	15	Usually not good	This is unusual	8	9
123	4	Network Time Protocol	Server to server will exhibit this behaviour	2	2

2.1.4 Top 10 Alert External Talkers

Out of the 3,232 systems that triggered the unknown snort rule set with an external source IP address Table 13 highlights the top 10. Over the next few tables ports that triggered alerts more than 100 times are in bold to give a sense of which ports where specifically

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

targeted by a given source, ports greater than 1023 are considered ephemeral ports and none refers to ICMP or fragmented packets, which don't specify a port.

Table 13 - Top 10 Alert External Talkers

External Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IP	Unique Dst Ports	Key Dst Ports (over 100 in bold)
134.192.42.11	21803	SOA (comm2.umaryland.edu.)	1	1	**51443**
131.92.177.18	5211	aeclt-cf00a4.apgea.army.mil.	1	1	**524**
209.164.32.205	4771	209.164.32.205.ptr.us.xo.net.	13	72	**none**, **0**, 80, 139, 113
68.55.155.26	3733	pcp05129829pcs.elkrdg01.md.comcast.net.	1	1	**8009**
69.136.228.63	3470	pcp08652049pcs.towson01.md.comcast.net.	1	1	**51443**
64.12.24.34	3076	SOA (dns-01.ns.aol.com.)	3	3	**1605**, **2718**, 1232
220.197.192.39	2613	?	181	3	**80**, 6129, 2745
69.138.77.62	2479	pcp08479849pcs.desoto01.md.comcast.net.	2	3	**3019**, **524**, 80
151.196.115.104	2454	pool-151-196-115-104.balt.east.verizon.net.	1	2	**524**, 3019
64.12.24.35	2335	SOA (dns-01.ns.aol.com.)	4	4	**1971**, 3706, 1214, 65535

2.1.5 Top 10 Alert Internal Targets

These top 10 hosts out of 1,208 where specifically targeted by an external IP address triggering the unknown snort rule set. The command used to create this table removes all lines with an internal source IP address ensuring only external sources are further analysed and then calculates the frequency of the targets. I expected only to see internal IP addresses as targets from external systems, but there where also external IP addresses as targets in the list (not represented in the top 10). This could indicate there where alerts generated by spoofed packets, a topic further explored in section 2.2.1.

Table 14 - Top 10 Alert Internal Targets

Targeted Internal IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IP	Unique Dst Ports	Key Dst Ports (over 100 in bold)
130.85.30.4	35322	lan2.umbc.edu.	313	1645	All ephemeral
130.85.30.3	15912	lan1.umbc.edu.	199	658	80, rest ephemeral
130.85.43.8	3437	SOA (UMBC3.umbc.edu.)	9	3	**65535**, 0, 6667
130.85.97.43	2160	ppp-043.dialup.umbc.edu.	2	27	**none**, **0**, 20, 6, 7, 9, 11, 12, 13
130.85.43.13	2123	SOA (UMBC3.umbc.edu.)	7	4	**6667**, 1863, 42936, 80
130.85.97.55	1811	ppp-055.dialup.umbc.edu.	2	21	**none**, **0**, 80, 53, 10, 11, 12, 13, 14, 15
130.85.69.232	1526	lib-69-232.pooled.umbc.edu.	10	10	**65535**, 0,80,69
130.85.153.81	1393	refweb17.libpub.umbc.edu.	4	3	**65535**, 0, 40
130.85.17.4	1282	c00040.umbc.edu.	31	29	80, rest ephemeral
130.85.17.3	1161	c00039.umbc.edu.	27	27	All ephemeral

2.1.6 Top 10 Alert Internal Talkers

Out of total 149 systems, this is the top 10 with a source IP address of the internal network.

Table 15 - Top 10 Alert Internal Talkers

Internal Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IP	Unique Dst Ports	Key Dst Ports (over 100 in bold)
130.85.43.8	3235	SOA (UMBC3.UMBC.EDU.)	8	2	**65535**, 137
130.85.11.4	3112	quarantine.UMBC.EDU.	55	2	**3111**, 65535
130.85.69.232	2992	lib-69-232.pooled.umbc.edu.	2	2	**65535**, 69
130.85.11.7	2510	dc2.ad.UMBC.EDU.	3	1	**137**
130.85.43.13	2125	SOA (UMBC3.umbc.edu.)	3	1	**65535**
130.85.153.81	884	refweb17.libpub.umbc.edu.	1	1	**65535**
130.85.150.44	632	illiad.lib.umbc.edu.	206	1	**137**
130.85.75.13	506	chpdm.umbc.edu.	157	1	**137**
130.85.150.198	435	pharos2.lib.umbc.edu.	161	1	**137**
130.85.70.156	245	henry.umbc.edu.	1	239	All ephemeral

2.1.7 Top 10 Alert External Targets

864 different external hosts where targeted by systems with an internal IP address triggering the unknown snort rule set, Table 16 highlights the top 10 external targets.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table 16 - Top 10 Alert External Targets

Targeted External IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IP	Unique Dst Ports	Key Dst Ports (over 100 in bold)
64.12.24.34	3070	SOA (dns-01.ns.aol.com.)	3	4	**1605**, **2718**, 1232, 1168
67.167.3.240	2991	c-67-167-3-240.client.comcast.net.	1	1	**2894**
210.120.128.117	2606	nis.dacom.co.kr.	2	1	137
64.12.24.35	2169	SOA (dns-01.ns.aol.com. hostmaster.aol.net.	3	4	**1971**, 3706, 1214, 3883
169.254.0.0	1529	SOA (prisoner.iana.org.) this is an APIPA address	4	1	**137**
169.254.25.129	1216	SOA (prisoner.iana.org.) this is an APIPA address	2	1	**137**
24.43.50.166	1197	CPE0010a4ebceb5-CM.cpe.net.cable.rogers.com.	11	3	**5900**, 1109, 1058
195.36.245.141	884	f01m-6-141.d3.club-internet.fr.	1	1	**1759**
200.199.135.134	295	SOA (ns2.telemar-ba.net.br.)	4	48	**137**, ephemeral
65.222.188.7	223	savgw.citizen.org.	1	1	**65535**

2.1.8 Top 10 Alert Types from External Hosts

From all the alerts generated by attackers there were 45 different types triggered by systems with an external source address, Table 17 shows the top 10.

Table 17 - Top 10 Alert Types from External Hosts

External Alert Type	Count
EXPLOIT x86 NOOP	38953
130.85.30.4 activity	35321
130.85.30.3 activity	15911
High port 65535 tcp - possible Red Worm - traffic	9177
Tiny Fragments - Possible Hostile Activity	4420

External Alert Type	Count
Null scan!	1938
RFB - Possible WinVNC - 010708-1	1158
NMAP TCP ping!	869
SUNRPC highport access!	254
connect to 515 from outside	250

2.1.9 Top 10 Alerts Types from Internal Hosts

From all the alerts generated Table 18 shows the top 10 triggered by attackers with an internal source address, there where 18 different types in total.

Table 18 - Top 10 Alert Types from Internal Hosts

Internal Alert Type	Count
High port 65535 tcp - possible Red Worm - traffic	10317
SMB Name Wildcard	8638
RFB - Possible WinVNC - 010708-1	1202
Possible trojan server activity	232
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	65

Internal Alert Type	Count
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	38
NIMDA - Attempt to execute cmd from campus host	34
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	23
TFTP - Internal UDP connection to external tftp server	21
High port 65535 udp - possible Red Worm - traffic	17

2.2 Alert Details

This section discusses some of the suspicious activity identified from the tables in section 2.1. Expansion of only a few alert types is provided within this analysis, as there is not enough time or space to investigate all the activity. Alerts with the highest count are not always of the highest concern, a host that keeps just under the radar and only triggers a couple of alerts could be your highest threat, looking at hosts generating low alert counts could be equally rewarding. Due to the quantity of alerts, analysis of this type has not been done, it is considered more beneficial in the short term to remove the high alerting false positives to ensure future reporting will provide more value.

Through out discussions in this section I have colour coded some of the tables to give a better idea as to what systems I think are of a greater priority to investigate.

Colour		Definition
Red		These systems are a priority for investigation
Orange		These systems should be investigated
Yellow		These systems are of potential concern
Green		These systems should be OK

2.2.1 Alert #1 – Spoofed Packets

The first thing that drew my attention to this category of alerts was 37 external destination addresses and no internal source addresses, unless packets are being spoofed, how can there be external destination addresses with no internal source addresses.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Port	Count	%	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
none	4463	3	?	2	26	18	0	37

There where two alert types without ports associated with the alert:

Tiny Fragments - Possible Hostile Activity	4420
ICMP SRC and DST outside network	43

The suspicious numbers come from the "`ICMP SRC and DST outside network`" rule, a quick search shows there where 37 unique destinations, and 21 unique sources. Then checking both the alert and scan data for spoofed source addresses revealed a few more in the alert data but none in the scan logs:

```
awk -F: '{print $5":"$7":"$4}' alert.misc3 | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | cut -d: -
   f 2- | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | cut -d: -f 2 | sort | uniq -c | sort -rn
```

IP address	Comments
80	TCP SRC and DST outside network
43	ICMP SRC and DST outside network
1	EXPLOIT x86 NOOP

There where a total of 72 unique destinations and 42 external sources address, here are the top 5 sources and destinations

```
awk -F: '{print $7":"$5}' alert.misc3 | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | cut -d: -f 2-
   | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | sort | uniq -c | sort -rn
awk -F: '{print $5":"$7}' alert.misc3 | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | cut -d: -f 2-
   | grep -v '130\.85\.[0-9]\+\.[0-9]\+' | sort | uniq -c | sort -rn
```

Table 19 - Top5 Source and Destination Addresses (Spoofed Packets)

Src IP address	Count	Comments
192.168.0.2	12	RFC 1918
172.161.154.148	10	ACA19A94.ipt.aol.com.
172.208.148.242	9	ACD094F2.ipt.aol.com.
192.168.2.117	6	RFC 1918
172.153.200.207	6	AC99C8CF.ipt.aol.com.

Dst IP address	Count	Comments
128.235.251.108	12	mailhost.njit.edu.
216.155.193.161	5	cs34.msg.dcn.yahoo.com.
207.46.107.88	5	baym-cs288.msgr.hotmail.com.
65.32.145.223	4	6532145hfc223.tampabay.rr.com.
168.95.192.1	4	hntp1.hinet.net.

There where six RFC 1918 addresses within the `192.168.0.X` and `192.168.2.X`. range, these could be from a system with its address manually configured, while there was one "Automatic Private IP Address"[136] `169.254.224.102`. APIPA addresses are automatically configured on windows based systems when there are no other addresses from other sources available to the system, i.e. via DHCP, manual etc. Another odd address was `61.77.112.240` owned by Korea Telecom, a very unusual address to find on the network.

```
inetnum:        61.72.0.0 - 61.77.255.255
netname:        KORNET
descr:          KOREA TELECOM
descr:          KOREA TELECOM Internet
Operating Center
country:        KR
```

The rest of the source addresses are within the America Online address space, a "`whois`" lookup reveals the two major blocks of addresses

```
OrgName:    America Online
OrgID:      AOL
NetRange:   172.128.0.0 - 172.191.255.255
CIDR:       172.128.0.0/10
```

```
OrgName:    America Online
OrgID:      AOL
NetRange:   172.192.0.0 - 172.211.255.255
CIDR:       172.192.0.0/12, 172.208.0.0/14
```

This traffic pattern could be from systems dialled into America Online and connected to the campus network at the same time, these hosts would have two routes to the Internet, and on occasions may chose the wrong route thus exposing an unusual source address. The FQDN for the IP addresses in Table 19 does not show the destinations as particularly unusual supporting this hypothesis.

Recommendations

Without a policy that does not allow a user to dial to the Internet while connected to the University network there is not much you can do. A security measure that should be

[136] What is Automatic Private IP Addressing (APIPA)? – http://www.duxcw.com/faq/network/autoip.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

implemented at all routers is ingress and egress filtering to limit the exposure the University has to being part of a DoS attack using spoofed addresses.

2.2.2 Alert #2 – Port 65535

Almost all the tables in section 2.1 show some activity regarding port 65535

Port	Count	%	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
65535	10093	8	[trojan] Adore worm, [trojan] RC1 trojan, [trojan] Sins	3	38	27	28	60

The 10,093 alerts break down like this

Count	Alert type
10046	High port 65535 tcp - possible Red Worm - traffic
46	High port 65535 udp - possible Red Worm - traffic
1	SMB Name Wildcard

This section will concentrate on the "High port 65535" rules that appear to trigger on any traffic having a source or destination port of 65535. Though port 65535 is a legitimate ephemeral port, it is still unusual to see this quantity of packets. Most operating systems have a configurable range of addresses used for dynamic allocation as source ports[137], port 65535 is the last port in this range for most of the common operating systems. Normally, an OS will cycle through the range of ephemeral ports only using 65535 when it reaches the end. With the traffic totalling 8% of alerts, there has to be a better explanation.

The dshield database[138] shows the interest in this port varies considerably from day to day but does not get anywhere near the levels reached by ports like 80 and 137. Additional information on this port indicates there are Trojans including the Adore worm[139], RC1 Trojan[140], and the Sins Trojan that use this port. Pete Storm states the Adore worm does not use UDP, this suggests these are related to something else. Table 20 shows the top 10 sources and destination ports triggering these alerts.

Table 20 - Top 10 "High port 65535 tcp" source and destination ports

Port	Src #	Dst #	comments
65535	9446	10046	-
2894	2991	1515	ABACUS-REMOTE
1971	2090	2259	NetOp School
1605	2050	2018	Salutation Manager (Salutation Protocol)
2718	1015	1048	PN REQUESTER 2
1759	884	1351	SPSS License Manager
25	252	514	SMTP
80	126	85	HTTP
1627	68	77	-
1979	64	64	-

Ports	Src #	Dst #	comments
65535	91	46	-
4672	8	10	-
6257	3	29	-
4692	3	3	-
61898	2	-	-
2025	2	-	-
62939	1	-	-
58367	1	-	-
54719	1	-	-
49076	1	-	-

Systems generating this alert are correlated with the scan data to identify any scan activity these hosts may have performed. Firstly, a list of the 152 IP addresses that triggered the "`High port 65535 xxx`" alert rule and their frequency is collected (42 where internal systems), each one is checked against the scan data looking for activity.

Internal Src	Alert #	dst IP	Scan #
130.85.6.7	2	2	-
130.85.12.6	8	8	55
130.85.12.7	12	5	1
130.85.24.20	81	1	-
130.85.24.33	6	1	-
130.85.24.34	34	17	-
130.85.24.44	10	6	-
130.85.24.74	32	3	-
130.85.25.12	10	2	-
130.85.25.10	227	2	-
130.85.25.66	47	8	3519

Internal Src	Alert #	dst IP	Scan #
130.85.25.70	23	8	17877
130.85.25.71	19	10	14126
130.85.25.72	22	8	11526
130.85.25.73	19	7	11486
130.85.29.3	4	2	-
130.85.34.11	69	6	-
130.85.34.14	32	7	154637
130.85.43.13	2125	3	5790
130.85.43.3	18	4	3625
130.85.43.4	166	10	2249
130.85.43.5	34	9	167

Internal Src	Alert #	dst IP	Scan #
130.85.60.38	1	1	-
130.85.69.214	1	1	169918
130.85.69.226	2	1	73049
130.85.69.232	2991	2	225716
130.85.70.164	100	5	2583
130.85.75.27	3	1	-
130.85.75.99	3	1	-
130.85.97.100	1	1	285
130.85.97.105	2	1	7328
130.85.97.183	3	1	2830
130.85.98.56	8	2	7501

[137] http://www.ncftpd.com/ncftpd/doc/misc/ephemeral_ports.html
[138] http://www.dshield.org/port_report.php?port=65535&recax=1&tarax=2&srcax=2&percent=N&days=70
[139] http://www.sans.org/y2k/adore.htm
[140] http://www.blackcode.com/trojans/details.php?id=1073

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

130.85.25.67		5	3	2983	130.85.43.8		3235	8	46542	130.85.111.34		8	9	29502
130.85.25.68		8	4	6182	130.85.53.31		3	2	-	130.85.153.81		884	1	829
130.85.25.69		29	9	14628	130.85.60.17		8	3	-	130.85.153.97		39	4	838

Out of the 108 external addresses triggering this alert only five where also see in the scan logs, these should be investigated further for scan activity.

External Source		Scan #
66.238.42.230		1732
68.218.142.172		44
200.51.38.2		523
203.15.51.51		34810
213.180.193.68		36546

A simple explanation for this traffic suggests it is related to the Adore worm, correlating with other peoples practicals tend to suggest this is the case.

Recommendations

Investigate the identified internal hosts further and verify they are not infected with the Adore worm, if they have been infected they should be disconnected from the network and cleaned before being reconnected.

2.2.3 Alert #3 – Port 137 ("SMB Name Wildcard" activity)

This discussion looks at "`SMB Name Wildcard`" activity.

Port	Count	%	Known traffic using this port number	Unique Alerts	Unique Ext Src IP	Unique Int Dest IP	Unique Int Src IP	Unique Ext Dest IP
137	8638	7	NETBIOS Name Service	2	0	1	62	639

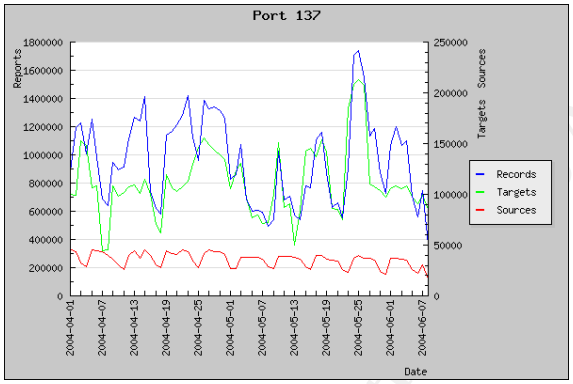


Figure 21 - DShield.org port 137 report (70-day trend)

These statistics show a lot of activity generated by port 137 directed to 639 different external addresses, almost one-tenth the quantity of internal source addresses. Looking at Figure 21 from the dshield [141] website we can see there is a consistently high level of activity associated with this port well known for Microsoft windows NETBIOS Name Service[142], also known for the quantity of vulnerabilities and Trojan horses that listen on this port.

Except for one, this traffic triggers the "`SMB Name Wildcard`" snort rule, which this section will concentrate on as we have already looked at the other alert type.

```
awk -F: '{print $8":"$4}' alert.misc3 | grep "^137:" | cut -d":" -f 2 | sort | uniq -c |
sort -rn
```

Alert Type	Count
SMB Name Wildcard	8637
High port 65535 tcp - possible Red Worm - traffic	1

Trying to identify the rule used by the University, we find this rule on Max Visions whitehats.com website, a possible snort rule looks like[143]:

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

This rule triggers on systems searching for resources on the target host by performing a wildcard query looking for its NetBIOS name table. This specific rule triggers on external

[141] http://www.dshield.org/port_report.php?port=137&recax=1&tarax=2&srcax=2&percent=N&days=70
[142] http://www.dshield.org/ports/port137.php
[143] http://www.whitehats.com/info/IDS177

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

hosts from any port targeting internal hosts on port 137, but none of the alerts show external source addresses.

```
awk -F: '{print $4":"$5":"$6}' alert.misc3 | grep "^SMB Name Wildcard" | awk -F: '{print $2":"$3}' | grep -v "^130\.85\.[0-9]\+\.[0-9]\+" | cut -d":" -f 2 | sort | uniq -c | sort -rn
```

Either the variables `$EXTERNAL` and `$INTERNAL` are the same and there was no traffic of this nature from outside the network or they have been customised to suit the Universities requirements. Looking for the destination ports related to "`SMB Name Wildcard`", we find in all but one case a destination port 137, which at this stage we will consider as an anomaly.

Destination Port	Count
137	8637
65535	1

The anomalous alert looks like:

```
04/22-02:30:14.905962:SMB Name Wildcard:130.85.11.4:1971:64.12.24.35:65535
```

The max vision link also provides insight into this traffic, the highlights of which are:

1. <This is> normal operation
2. Used when file sharing is active to determine NetBIOS names
3. When originating from an external network, is usually a pre-attack probe.
4. Information available includes:
   a. The NetBIOS name of the server.
   b. The Windows NT workgroup domain name.
   c. Login names of users who are logged into the server.
   d. Name of the administrator account if logged into the server.
5. It is considered best practice to ensure users outside of your network are not permitted to access the NetBIOS name service.

So far, we can consider these probes as benign, checking the source ports will help classify this traffic, this table shows the top 10 source ports and the percentage against the total SMB alert count.

```
awk -F: '{print $4":"$5":"$6}' alert.misc3 | grep "^SMB Name Wildcard" | awk -F: '{print $2":"$3}' | grep "^130\.85\.[0-9]\+\.[0-9]\+" | cut -d":" -f 2 | sort | uniq -c | sort -rn | head
```

Source Port	Count	%
137	7395	87
1059	235	3
1072	187	2
1060	120	1
1050	102	1

Source Port	Count	%
1082	99	1
1058	71	<1
1131	56	<1
1109	42	<1
1044	16	<1

We are potentially seeing two different types of traffic[144]. Traffic with a source and destination port 137 is probably benign, and is normal windows traffic touched on again in Alert #4 – Reflexive Ports. Traffic with a source port above 1023 is more likely to be the "Opaserv" or "BugBear" worms. Systems generating traffic with a source port not 137 could be infected with these worms and should be checked for signs of infection using an up to date virus scanner, here are these hosts:

```
awk -F: '{print $4":"$6":"$5}' alert.misc3 | grep "^SMB Name Wildcard" | awk -F: '{print $2":"$3}' | grep -v "^137:" | cut -d":" -f 2 | sort | uniq -c | sort -rn
```

Source Port		Count of alerts with source >1023
130.85.150.44		515
130.85.150.198		380
130.85.43.14		118
130.85.43.16		98

Source Port		Count of alerts with source >1023
130.85.43.5		81
130.85.43.17		46
130.85.66.15		3
130.85.11.4		2

Recommendations

The 137 to 137 traffic looks legitimate but the traffic with a source port greater than 1023 is potentially serious, this traffic should be further scrutinised. This rule is also only of value as additional information during an investigation, it has many false positives. A snort rule

[144] http://www.dslreports.com/forum/remark,5995337~mode=flat

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

that looks for traffic that has a source not 137 could be more valuable as an indicator of nefarious activity as identified in the previous link.

2.2.4 Alert #4 – Reflexive Ports

In all but a few cases packets discovered with the same source and destination ports are a sign of packet crafting, almost certainly unexpected, and usually up to no good. This section will look at the legitimacy of the ports identified in Table 12 and reveal any hosts generating suspicious traffic.

Port 137: According to Building Internet Firewalls[145]:

> Microsoft implementations use port 137 for queries as well as responses.

Indicating this traffic is more likely to be legitimate, if there are rules that need to trigger on traffic with this pattern they need to be more specific in what content they are looking for to reduce the rate of false positives. This traffic was discussed in detail in section 2.2.3.

Port 0: A packet containing port 0 as either a source or a destination is not normal, for it to appear as a reflexive port makes it a real concern. Here are the top10 source and destination IP addresses, there where no internal systems generating this traffic and the systems in bold also appear in the Port 0 table under section 2.4.2

```
awk -F: '{print $6":"$8":"$5":"$4}' alert.misc3 | grep "^0:0:" | cut -d":" -f 3,4 | sort |
uniq -c | sort -rn
awk -F: '{print $6":"$8":"$7":"$4}' alert.misc3 | grep "^0:0:" | cut -d":" -f 3,4 | sort |
uniq -c | sort -rn
```

Source IP	#	Alert	FQDN
61.48.8.56	614	Null scan!	SOA (ns1.apnic.net.)
82.83.43.1	354	Null scan!	dsl-082-083-043-001.arcor-ip.net.
209.164.32.205	314	Null scan!	209.164.32.205.ptr.us.xo.net.
203.210.158.107	48	Null scan!	localhost.
83.28.245.202	40	Null scan!	bmz202.neoplus.adsl.tpnet.pl.
217.95.226.63	37	Incomplete Packet Fragments Discarded	bmz202.neoplus.adsl.tpnet.pl.
68.218.142.172	30	Null scan!	adsl-218-142-172.jax.bellsouth.net.
213.199.82.204	22	Null scan!	204.net82.skekraft.net.
217.31.166.66	18	Null scan!	schismatrix.gnulix.org.
211.121.26.205	15	Null scan!	SDDfa-03p4-205.ppp11.odn.ad.jp.

Destination IP	#	Alert
130.85.112.209	614	Null scan!
130.85.82.109	354	Null scan!
130.85.97.43	139	Null scan!
130.85.12.6	134	Null scan!
130.85.97.55	114	Null scan!
130.85.81.116	55	Null scan!
130.85.70.164	49	Incomplete Packet Fragments Discarded
130.85.153.97	46	Null scan!
130.85.153.81	41	Null scan!
130.85.70.72	40	Null scan!

This traffic is diffidently not normal and can be used to fingerprint the target OS[146]. Darrin Wassom[147] did an analysis on port 0 traffic where he suggests similar traffic is crafted in an attempt to fingerprint targeted systems. As there where no internal addresses acting as the source the only recommendation is to investigate the external sources for signs of further scan activity and if found report it to the system owner.

Port 53: According to Building Internet Firewalls[148]:

> A DNS server uses well-known port 53 as its server port for TCP and UDP. It uses a port above 1023 for TCP requests. Some servers use 53 as a source port for UDP requests, while others will use a port above 1023. A DNS client uses a random port above 1023 for both UDP and TCP.

Identifying the source and destinations may reveal the legitimacy of these connections.

Source IP	#	FQDN
64.152.70.68	102	proximitycheck2.allmusic.com.
63.211.17.228	101	proximitycheck1.allmusic.com.
219.133.41.254	2	SOA (dns.guangzhou.gd.cn.)
209.135.37.205	2	msrbaa03.usi.net.
200.52.107.160	1	np1mty05.banamex.com.

Destination IP	#	FQDN
130.85.1.3	188	UMBC3.UMBC.EDU.
130.85.1.4	20	UMBC4.UMBC.EDU.

[145] http://secu.zzu.edu.cn/book/NetWork/NetworkingBookshelf_2ndEd/fire/ch20_03.htm
[146] http://www.securiteam.com/securityreviews/5XP0Q2AAKS.html
[147] http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00002.html
[148] http://secu.zzu.edu.cn/book/NetWork/NetworkingBookshelf_2ndEd/fire/ch20_01.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The only alert triggered by this traffic is "NMAP TCP ping!" an example of a possible rule[149]:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP"; ack:0; flags:A,12;
flow:stateless; reference:arachnids,28; classtype:attempted-recon; sid:628; rev:7;)
```

This rule is looking for TCP traffic that has the ack number set to zero, TCP is a legitimate transport protocol for DNS in two cases, during a zone transfer, or when a UDP packet is too small to return all the requested information back to the requester.

The destination systems are DNS servers, that looks OK, the two systems generating the most traffic are quite interesting, details are discussed in a post by Ashley Thomas[150] to the incidents.org mailing list. In short, these are signs of a Radware system performing a proximity check for the location of a host, presumably a system internal to the University is looking for something on the allmusic.com website. These Radware systems are attempting to improve the clients experience by trying to locate them in the assumption they are within close proximity to their DNS servers.

The other sources could be older DNS servers that still use source port 53.

Port 25: According to Building Internet Firewalls,[151] there is no reason for this type of traffic:

```
SMTP is a TCP-based service. SMTP receivers use port 25. SMTP senders use
a randomly selected port above 1023.
```

Source IP	#	FQDN
63.84.193.227	133	mx2.ezinedirector.net.
63.84.193.228	45	mx3.ezinedirector.net.
66.77.190.140	7	sitemail1.fanball.com.
66.63.162.42	6	ems8.emsemail.biz.

Destination IP	#	FQDN
130.85.12.6	191	mxin.umbc.edu.

"TCP SMTP Source Port traffic" is the only alert type triggered by this traffic. This table indicates the traffic is coming from mail servers directing traffic to the internal University Mail server. This looks OK to me except SMTP is not suppose to use source port 25, Mark Menke[152] put this traffic down to a false positive and Sidney Faber[153] recognised the traffic as abnormal but is uncertain as to what is happening. I was also unable to locate any reason for this traffic except to suggest that a Trojan that exhibits this traffic pattern could have compromised these systems or they use a mailer that goes against standards.

Port 80: There is no reason for this traffic according to Building Internet Firewalls[154] "NMAP TCP ping!" is the only alert triggered by this traffic:

```
HTTP is a TCP-based service. Clients use random ports above 1023. Most
servers use port 80, but some don't.
```

Source IP	#	FQDN
211.152.3.40	12	SOA (ns.cnc.ac.cn.)
159.226.208.40	11	SOA (ns.cnc.ac.cn.)
202.28.21.2	6	SOA (ns.thnic.net.)
212.199.61.6	4	212.199.61.6.forward.012.net.il.
212.179.16.106	3	bzq-179-16-106.cust.bezeqint.net.
210.202.193.1	3	richtek.com.
209.6.58.139	3	rcn.berklee.edu.
204.96.18.6	3	204.96.18.6.cust.phila.sprintlink.net.
203.146.247.2	3	SOA (ns.tnet.co.th.)
193.144.127.9	3	SOA (ninot.gva.es.)

Destination IP	#	FQDN
130.85.24.44	30	userpages.umbc.edu.
130.85.34.11	16	SOA (UMBC3.UMBC.EDU.)
130.85.6.7	11	umbc7.umbc.edu.
130.85.24.34	11	www.umbc.edu.
130.85.18.62	10	SOA (UMBC3.UMBC.EDU.)
130.85.7.114	5	SOA (UMBC3.UMBC.EDU.)
130.85.2.224	4	SOA (UMBC3.umbc.edu.)
130.85.156.62	4	SOA (UMBC3.UMBC.EDU.)
130.85.109.175	4	SOA (UMBC3.umbc.edu.)
130.85.82.50	3	oit-82-50.pooled.umbc.edu.

There where 83 sources all external with this traffic pattern, the Code Red worm fits this pattern[155] putting these systems on the list for further investigation for scan activity and informing the system owners.

---

[149] http://www.snort.org/snort-db/sid.html?sid=628
[150] http://www.dshield.org/pipermail/intrusions/2002-December/006176.php
[151] http://secu.zzu.edu.cn/book/NetWork/NetworkingBookshelf_2ndEd/fire/ch16_02.htm
[152] http://www.giac.org/practical/Mark_Menke_GCIA.doc
[153] http://www.giac.org/practical/Sidney_Faber_gcia.doc
[154] http://secu.zzu.edu.cn/book/NetWork/NetworkingBookshelf_2ndEd/fire/ch15_03.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Port 65535: This traffic type is discussed in Alert #2 – Port 65535

Port 123: Building Internet Firewalls[156] suggests there is a legitimate case for this traffic:

```
client-to-server query - Source port above 1023, destination port 123
server-to-client response - Source port 123, destination port above 1023
server-to-server query or response - Source and destination ports both 123
```

Source IP	#	FQDN
66.250.188.13	3	66.250.188.13.chaincast.com.
212.204.214.70	1	dist001.media.widexs.nl.

Destination IP	#	FQDN
130.85.66.29	3	SOA (UMBC3.umbc.edu.)
130.85.97.50	1	ppp-050.dialup.umbc.edu.

"EXPLOIT NTPDX buffer overflow"[157] is the only alert triggered by this traffic, this rule triggers on UDP packets with a payload greater than 128-bytes, this is the only test done on the traffic so there is a chance for false positive. Verification of the destination hosts for compromise should be done as this exploit allows a root compromise, though the identity of the dial up host may need a little extra work to identify. These IP addresses where not seen in the scan data targeting port 123 (see section 2.4.2) or any other systems. The registration information on these two hosts is in sections 3.4 and 3.5.

Recommendations

Investigate the identified hosts in this section for possible compromise.

2.2.5 Alert #5 – Exploit x86 Set(uid/gid) 0

At first glance, these alerts are of concern as they could indicate a compromise.

Message	No.	%
EXPLOIT x86 setuid 0	28	<1
EXPLOIT x86 setgid 0	26	<1

Snort rules that may match these alerts[158] [159]:

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 setuid 0";
   content:"|B0 17 CD 80|"; reference:arachnids,436; classtype:system-call-detect; sid:650;
   rev:8;)
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 setgid 0";
   content:"|B0 B5 CD 80|"; reference:arachnids,284; classtype:system-call-detect; sid:649;
   rev:8;)
```

The unknown rule set used by the University seems to be heavily influenced by rules on Max Visions site, so the rules are more likely to be[160] [161]:

```
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS283/shellcode_shellcode-x86-setuid0";
   flags: A+; content: "|b017 cd80|"; classtype: system-attempt; reference: arachnids,283;)
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS284/shellcode_shellcode-x86-setgid0";
   flags: A+; content: "|b0b5 cd80|"; classtype: system-attempt; reference: arachnids,284;)
```

The snort specific rules trigger on traffic from an external IP address and port as defined by the $SHELLCODE_PORTS variable to an internal IP address on any port. While the Max Vision rules will trigger on traffic from an external IP address with any port to an internal IP address on any port. Some things we can see from the traffic to identify the likely rules.

1. There are no alerts by traffic originating from the internal network, this could also indicate the variables for $INTERNAL and $EXTERNAL have been tunned to reduce the likelihood of false positives.

```
awk -F: '{print $4":"$5":"$6}' alert.misc3 | grep "EXPLOIT x86 setuid 0" | awk -F: '{print
   $2":"$3}' | grep "^130\.85\.[0-9]\+\.[0-9]\+" | cut -d":" -f 2 | sort | uniq -c | sort -
   rn
awk -F: '{print $4":"$5":"$6}' alert.misc3 | grep "EXPLOIT x86 setgid 0" | awk -F: '{print
   $2":"$3}' | grep "^130\.85\.[0-9]\+\.[0-9]\+" | cut -d":" -f 2 | sort | uniq -c | sort -
   rn
```

---

[155] http://www.dshield.org/pipermail/intrusions/2001-July/000986.php
[156] http://secu.zzu.edu.cn/book/NetWork/NetworkingBookshelf_2ndEd/fire/ch22_05.htm
[157] Potential rule triggered http://www.digitaltrust.it/arachnids/IDS492/event.html
[158] http://www.snort.org/snort-db/sid.html?sid=650
[159] http://www.snort.org/snort-db/sid.html?sid=649
[160] http://www.whitehats.com/info/IDS283
[161] http://www.whitehats.com/info/IDS284

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2. In a default snort rule we find the `$SHELLCODE_PORTS` variable set to anything but port 80 "`!80`". Checking the ports we find traffic on port 80 did trigger this indicating the `$SHELLCODE_PORTS` variable is either set to "`any`" or the rule does not use this variable. This makes the Max Vision rule the likely candidate.

Top 5 source and destination ports for both alerts:

Setuid Source ports	
Port (setuid)	Count
80	2
3476	2
62622	1
55122	1
52895	1

Setgid Source ports	
Port (setgid)	Count
80	10
9254	1
6881	1
52551	1
43559	1

Setuid Destination ports	
Port (setuid)	Count
119	5
6882	2
6881	2
3250	2
2894	2

Setgid Destination ports	
Port (setgid)	Count
4497	2
119	2
6885	1
6882	1
6881	1

Max Vision reports what the possibilities of false positives are for IDS283, there is almost identical information regarding IDS284, the highlights are:

1. This signature is very short
2. [it] does not contain context clues to reduce false positives
3. false alarms [could occur] where binary data is transferred from outside the network [examples are]
   a. A user downloading binary files from an external web server.
   b. binary data streams is Napster mp3 transfers

Source port 80 plays a significant part in triggering this rule, this traffic is likely to be clients downloading files from web sites containing a sequence of hex character "`B017 CD80`" or :"`B0B5 CD80`"

Recommendations

Due to the likelihood of false positive, a few options should be considered:

1. Turn off this rule;
   These rules are too generic and highly likely to trigger false positives.
2. Tune the rule to be more specific;
   The snort rule reduces the false positive rate by not inspecting traffic for port 80, as discussed by Max Vision there is other traffic that can trigger a false positive. A better solution would be to improve the content being searched to identify the specific attack, the downside here is, and when you get too specific, you create the potential for a false negative.
3. Only use it for further evidence when investigating a compromise.
   If a more reliable rule triggered indicating a possible break in, when you cross-reference the IP address for a history of alerts from that address, and this rule came up, there is a higher chance it is not a false positive.

2.2.6 Alert #6 – 130.85.30.4 and 130.85.30.3 activity

These alert types are mentioned in previous practicals, a correlation of information from other analysts may prove interesting. It has been suggested these rules trigger on any traffic directed to these systems[162], the triggered traffic is certainly one way, with no reply traffic recorded removing the possibility to determine a sequence of events.

Message	FQDN of system	No.	%
130.85.30.4 activity	lan2.umbc.edu.	35321	27
130.85.30.3 activity	lan1.umbc.edu.	15911	12

Some possible reasons these hosts are specifically targeted by a snort rule include:

1. They are Honeypots and the University wants to keep a record of all packets targeting the systems (though in this case logging is less than adequate);
2. They are valuable systems containing crucial information; or

[162] http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf (page 49)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3. These rules are used to produce reports detailing their usage for back charging etc.

Top 10 destination ports on to both systems show these systems are presumably related to University activity, these ports as noted by Pete Storm seem to correlate with a Novell Netware system[163] [164] [165]:

Table 21 - Top 10 Ports Targeted on 130.85.30.3

Dst Port	Count	%	Comments
524	13748		NetWare Core Protocol (NCP)
3019	1393		NDPS RMS
80	528		GroupWise Web Access, iFolder, iMonitor, NetWare Enterprise Web Server, NetWare Remote Manager (NRM), NetWare Web Access, Novell Internet Messaging System (NIMS) , Portal Services, Zenworks for Servers 2
2745	74		Beagle.C through Beagle.K (This could just be related to scan activity) see section 2.4.6
6129	65		Dameware Remote Admin (This could be related to scan activity)
443	47		BorderManager GroupWise Web Access, iFolder iPrint, NetWare Enterprise Web Server, Novell Internet Messaging System (NIMS), Portal Services, Zenworks for Servers 2
4899	11		Remote Administrator default port (This could just be related to scan activity)
8009	6		NetWare Remote Manager (NRM), Zenworks for Servers 2
4000	5		Could be related to scan activity see section 2.4.6
427	4		SLP

Table 22 - Top 10 Ports Targeted on 130.85.30.4

Dst Port	Count	%	Comments
51443	28792		Optional alternate port for iFolder Management Console[166]
8009	3780		NetWare Remote Manager (NRM), Zenworks for Servers 2
80	1365		GroupWise Web Access, iFolder, iMonitor, NetWare Enterprise Web Server, NetWare Remote Manager (NRM), NetWare Web Access, Novell Internet Messaging System (NIMS), Portal Services, Zenworks for Servers 2
524	1198		NetWare Core Protocol (NCP)
2745	61		Beagle.C through Beagle.K (This could just be related to scan activity) see section 2.4.6
6129	50		Dameware Remote Admin (This could be related to scan activity)
443	29		BorderManager GroupWise Web Access, iFolder iPrint, NetWare Enterprise Web Server, Novell Internet Messaging System (NIMS), Portal Services, Zenworks for Servers 2
4899	9		Remote Administrator default port (This could just be related to scan activity)
20168	6		possibly related to the Lovgate virus (This could just be related to scan activity)
4000	5		Could be related to scan activity see section 2.4.6

Only the first few ports look sensible, the rest are probably related to scan activity, the sources of this scan activity should be investigated further to determine the purpose of this activity. The top 10 source address targeting each system:

Table 23- Top 10 Source Addresses Targeting 130.85.30.3

IP address	Count	%	FQDN
131.92.177.18	5211	33	aeclt-cf00a4.apgea.army.mil.
69.138.77.62	2458	15	pcp08479849pcs.desoto01.md.comcast.net.
151.196.115.104	2454	15	pool-151-196-115-104.balt.east.verizon.net.
68.34.94.70	1901	12	TSU-68-34-94-70.tsu01.md.comcast.net.
68.55.113.28	632	4	pcp311377pcs.woodln01.md.comcast.net.
68.57.90.146	629	4	pcp912734pcs.brndml01.va.comcast.net.
68.55.27.157	617	4	pcp02560368pcs.owngsm01.md.comcast.net.
68.55.250.229	390	2	pcp261188pcs.howard01.md.comcast.net.
141.157.40.149	287	2	pool-141-157-40-149.balt.east.verizon.net.
138.88.98.71	227	1	pool-138-88-98-71.res.east.verizon.net.

Table 24 - Top 10 Source Addresses Targeting 130.85.30.4

IP address	Count	%	FQDN
134.192.42.11	21803	62	SOA (comm2.umaryland.edu.)
68.55.155.26	3733	11	pcp05129829pcs.elkrdg01.md.comcast.net.
69.136.228.63	3470	10	pcp08652049pcs.towson01.md.comcast.net.
68.33.49.146	1598	5	pcp03625900pcs.mtromd01.md.comcast.net.
172.209.111.241	827	2	ACD16FF1.ipt.aol.com.
68.55.191.197	417	1	pcp05510211pcs.owngsm01.md.comcast.net.
68.55.113.28	400	1	pcp311377pcs.woodln01.md.comcast.net.
68.54.168.204	305	<1	pcp02772508pcs.howard01.md.comcast.net.
68.34.94.70	292	<1	TSU-68-34-94-70.tsu01.md.comcast.net.
64.8.195.10	228	<1	64-8-195-10.client.dsl.net.

With a list of the 197 source addresses targeting 130.85.30.3 and the 313 addresses targeting 130.85.30.4 the scan data was checked for these same addresses. This table

[163] Netware 6 - http://www.novell.com/documentation/nw6p/index.html?page=/documentation/lg/nw6p/adminenu/data/aclkn27.html
[164] Netware 6 - http://support.novell.com/cgi-bin/search/searchtid.cgi?/10065719.htm
[165] Netware 5 and earlier - http://support.novell.com/cgi-bin/search/searchtid.cgi?/10014320.htm
[166] http://www.novell.com/documentation/ifolder21/index.html?page=/documentation/ifolder21/readme/data/ahf1v06.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

shows the attackers that also appear in the scan data targeting both systems. These systems need further investigation to determine if they did indeed perform some type of unwelcome scan activity and then reported to the system owner to have the activity stoped.

Table 25 – Source Addresses Seen in the Scan logs who targeted both 130.85.30.3 and 130.85.30.4

IP address		Alert #	Scan #	S IP #
24.222.128.240		2	10979	8808
24.43.50.166		3	4818	3770
24.6.74.41		2	27156	15462
24.97.20.62		2	13435	10473
61.220.43.174		2	25411	15015
64.136.199.197		2	20487	11182
64.163.92.253		2	18561	12889
64.68.188.105		2	20178	13374
65.82.70.5		2	14883	11119
66.205.222.145		2	6246	6100
67.161.192.207		2	23745	14823
67.36.69.73		2	15371	11570
67.92.156.144		2	13212	10261
80.24.133.184		2	8341	7032

IP address		Alert #	Scan #	IP #
80.63.67.86		2	7845	6647
159.218.66.88		3	47	12
165.246.35.168		2	26654	15477
193.120.129.82		2	14580	10979
195.57.122.14		2	12594	9883
201.128.103.27		2	3212	3164
202.183.231.196		2	5955	5738
202.60.240.4		2	12381	9616
204.116.105.26		2	12353	9712
206.222.14.209		2	16732	11957
207.178.156.104		2	7901	6753
208.182.190.91		2	8243	6399
210.109.146.30		2	16215	11849
211.136.159.206		2	7293	5622

IP address		Alert #	Scan #	IP #
212.192.241.20		3	4886	4451
212.29.89.240		2	12978	9899
212.3.250.197		2	9054	7721
213.186.239.244		2	6575	5802
213.189.229.5		2	18527	12925
213.39.228.182		2	13907	10723
213.73.245.83		2	13196	10265
216.118.120.35		2	12845	9858
216.199.191.118		2	9123	7587
217.37.163.109		2	14080	10786
217.37.21.213		2	6465	6008
218.156.65.55		2	17827	12365

Table 25 highlights the value these rules can provide in detecting scan activity, we can see in all but three cases it was only the two host specific rules that fired. Once cross-referenced with the number of entries in the scan data and the number of unique destination IP addresses by that host, all are revealed as network scanners. Further investigation might suggest the intent behind these scans, i.e. worm activity, an attacker trying to enumerate available services on the University network etc.

Table 26 and Table 27 show the attackers who only triggered one or other host specific rule, this reveals further scan activity and in most cases is a single host targeting thousands of University hosts. The hosts that triggered more than one alert in this case are very suspicious, and certainly require further investigation:

Table 26 - Attackers seen in the scan logs only targeting 130.85.30.3

IP address		Alert #	Scan #	IP #
61.30.107.50		1	5081	4892
66.69.180.216		1	17820	12706
68.157.167.58		1	11187	8941
68.219.142.66		1	8840	7608
81.74.150.162		1	6718	5835
207.3.145.130		5	18380	11027
207.96.102.27		1	9986	8125

IP address		Alert #	Scan #	IP #
208.177.178.20		1	28	28
209.90.101.77		1	12529	9276
210.23.197.163		1	4212	4127
218.155.244.116		2	29	7
218.188.13.228		1	5801	5800
220.197.192.39		3	31604	6629

Table 27 - Attackers seen in the scan logs only targeting 130.85.30.4

IP address		Alert #	Scan #	IP #
24.218.87.53		1	1592	1560
61.248.203.170		1	6040	5386
61.84.6.89		2	162	46
64.166.194.201		1	10389	8381
66.53.128.107		1	1087	1087
80.191.163.12		1	19073	13220
80.38.233.3		1	13892	10698
130.206.173.195		1	8961	7767

IP address		Alert #	Scan #	IP #
130.39.31.182		1	10757	8764
136.145.193.14		1	13005	10145
144.92.91.95		1	11075	8905
202.30.111.21		2	51	11
203.106.94.2		1	5363	4706
209.115.211.86		1	7904	6844
212.93.134.12		1	5943	5710
218.154.28.67		2	35	7

Recommendations

Because the University is logging every packet directed to these systems these rules offer great opportunities during the investigation process, some uses for these rules include:

1. To identify unusual ports that have been targeted on the network;
   The ports listening on these systems are well known, any systems targeting ports outside these should be further investigated.
2. Identify scanning systems.
   As demonstrated, we are able to determine systems that have been scanning the network by just correlating the alert data with the scan logs. This revealed systems that have been performing other activity on the campus network.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

These systems almost work like a honeypot in that traffic outside the expected is logged providing the necessary first lead for further investigation. It is recommended that the University continue to use these techniques to investigate network activity.

2.3 Scans Summary

The tables in this section display the alert data in various ways helping identify suspicious activity, some of which is further discussed in section 2.4

2.3.1 Top 20 Scan Types

From the 13,816,238 scan entries, there where 236 different types of scan activity identified within the chosen timeframe, Table 28 shows the 20 most triggered scan types and the percentage of the total.

Table 28 - Top 20 Scan Types

Type	Flags	No.	%
SYN	******S*	8252100	60
UDP		5542631	40
FIN	*******F	9866	<0.1
SYN	12****S* RESERVEDBITS	7059	<0.1
NULL	********	913	<0.1
UNKNOWN	12*A**S* RESERVEDBITS	631	<0.1
UNKNOWN	1**A**S* RESERVEDBITS	580	<0.1
UNKNOWN	*2*A**S* RESERVEDBITS	489	<0.1
INVALIDACK	***A*R*F	432	<0.1
VECNA	****P***	150	<0.1

Type	Flags	No.	%
NOACK	**U*P*S*	135	<0.1
NOACK	**U**RSF	113	<0.1
NOACK	**U**RS*	109	<0.1
UNKNOWN	1****R** RESERVEDBITS	67	<0.1
VECNA	**U*P***	58	<0.1
UNKNOWN	*2***R** RESERVEDBITS	40	<0.1
UNKNOWN	12***R** RESERVEDBITS	33	<0.1
XMAS	**U*P**F	30	<0.1
INVALIDACK	*2*A*R*F RESERVEDBITS	25	<0.1
UNKNOWN	*2*A**** RESERVEDBITS	24	<0.1

2.3.2 Top 20 Scan Destination Ports

Table 29 shows the top 20 destination ports out of 234 observed that triggered the scan detection engine within the selected period.

Table 29 - Top 20 Scan Destination Ports

Port	Count	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
53	4681210	Domain Name Server	13	66	14	6	115665
135	2616043	DCE endpoint resolution, NCS local location broker	8	124	263	20	1844430
2745	786716	Urbis geolocation service, Bagle Virus Backdoor	8	41	6272	25	570176
6129	656848	Dameware Remote Admin	9	54	15727	20	364557
80	604263	World Wide Web HTTP	21	255	15496	146	320430
445	582272	Win2k+ Server Message Block	4	21	259	15	428583
1025	570135	network blackjack, [trojan] Fraggle Rock, listener RFS remote_file_sharing, [trojan] NetSpy, [trojan] Remote Storm	3	4	6	25	413812
3127	514793	W32/MyDoom, W32.Novarg.A backdoor, MyDoom.C / Doomjuice	6	14	164	24	377757
139	473811	NETBIOS Session Service, [trojan] SMB Relay, [trojan] Sadmind	6	30	259	19	346566
3410	316638	Backdoor.OptixPro.13	4	4	4	21	244707
443	297002	HTTP protocol over TLS SSL	11	42	15725	29	33
25	249438	Simple Mail Transfer	15	320	12372	18	6074
5000	141574	[trojan] Back Door Setup, Universal Plug and Play	3	5	5	19	114368
4899	70007	Remote Administrator default port	2	12	15584	2	1162
6346	43846	gnutella-svc, BearShare file sharing app	6	742	37	41	10650
20168	41026	possibly related to the Lovgate virus	2	5	15111	1	2
4000	29522	[trojan] Connect-Back Backdoor, [trojan] SkyDance, w32.witty.worm	8	9	13608	4	9
41170	26774	Could be "Blubster" a file sharing program.	2	1	1	8	9613
32773	25862	Sun's rpc.nisd program	2	2	13877	2	1
21	23872	File Transfer [Control]	5	21	13527	17	21

2.3.3 Scan Reflexive Ports and IP Addresses

With a few exceptions, it is uncommon to see source and destination ports the same, it is certainly an issue when source and destination IP addresses are the same. There where no reflexive IP addresses in the Scan data, but there where 33 reflexive ports, the top 10 are detailed in Table 30 and are potentially worth further investigation.

Table 30 - Scan Reflexive Port Combinations

Port	Count	Known use.	Comments	Src IP #	Dst IP #
123	18870	Network Time Protocol	Server to server will exhibit this behaviour	3	53

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4672	13557	remote file access server	Needs investigation	4	9169
0	1226	Reserved, many vulnerabilities	Port 0 is unusual at the best of times	80	50
6112	871	CDE subprocess control, FSGS (game)	Needs investigation	7	304
6346	550	gnutella-svc, BearShare file sharing app	Needs investigation	11	520
2745	196	Urbis geolocation service, Bagle Virus Backdoor	Needs investigation	11	145
3127	122	W32/MyDoom, W32.Novarg.A backdoor	Needs investigation	11	94
3531	115	?	Needs investigation	11	46
5000	74	[trojan] Back Door Setup, [trojan] BioNet Lite, Free Internet Chess Server & more	Needs investigation	5	10
3410	70	?	Needs investigation	10	55

2.3.4 Top 10 Scan External Talkers

Table 31 highlights the top 10 systems out of 2,090 with an external source IP address that performed some form of scanning.

Table 31 - Top 10 Scan External Talkers

External Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IPs	Unique Dst Ports	Key Dst Ports (over 10,000 in bold)
213.180.193.68	36546	proxychecker.yandex.net.	5	30488	80, 81, and on and on, mostly ephemeral ports
203.15.51.51	34810	CNAME 51.32-63.51.15.203.in-addr.arpa. PTR tester-51-2.sorbs.net	1	33090	No observed pattern, most are ephemeral ports.
220.197.192.39	31604	?	6629	8	**2745**, **6129**, 80, 139, 3127, 53, 1025, 1981
24.6.74.41	27156	c-24-6-74-41.client.comcast.net.	15462	1	**443**
165.246.35.168	26654	?	15477	1	**6129**
61.220.43.174	25411	61-220-43-174.HINET-IP.hinet.net.	15015	1	**443**
67.161.192.207	23745	c-67-161-192-207.client.comcast.net.	14823	1	**443**
64.136.199.197	20487	64-136-199-197-dhcp-kc.everestkc.net.	11182	2	**80**, 443
64.68.188.105	20178	SOA (ns1.nvc.net. hostmaster.nvc.net.)	13374	1	**443**
80.191.163.12	19073	SOA (ns.ripe.net. ops-80.ripe.net.)	13220	1	**6129**

2.3.5 Top 10 Scan Internal Targets

Table 32 shows the top 10 systems out of 15,754 with a destination IP address of an internal host that received scan type activity.

Table 32 - Top 10 Scan Internal Targets

Internal Destination IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IPs	Unique Dest Ports	Key Dest Ports (over 100 in bold)
130.85.27.232	35015	SOA (UMBC3.umbc.edu.)	52	33138	80, 443, 6129
130.85.97.65	22027	ppp-065.dialup.umbc.edu.	54	21586	443, 6129
130.85.97.159	14535	ppp-159.dialup.umbc.edu.	40	13214	443, 6129
130.85.66.30	4960	SOA (UMBC3.umbc.edu.)	56	4759	80, 113, 443, 6129
130.85.12.6	2663	mxin.umbc.edu.	359	50	0, **25**, 80, 443, 6129
130.85.6.7	2073	umbc7.umbc.edu.	67	16	80, **110**, 443, 6129, 4899
130.85.97.93	1817	ppp-093.dialup.umbc.edu.	140	14	443, 6129, **6346**
130.85.18.25	1798	shsfrontpc-02.umbc.edu.	49	1037	**0**, 53, 80, 443, **2710**, 2745, 6129, 7001
130.85.112.204	1796	SOA (UMBC3.UMBC.EDU.)	311	12	443, 6129, **6346**
130.85.97.73	703	ppp-073.dialup.umbc.edu.	50	633	80, 443, 6129

2.3.6 Top 10 Scan Internal Talkers

Table 33 shows the top 10 systems out of 208 with an internal source IP address that performed scan type activity.

Table 33 - Top 10 Scan Internal Talkers

Internal Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IPs	Unique Dst Ports	Key Dst Ports (over 100,000 in bold)
130.85.1.3	3630346	UMBC3.UMBC.EDU.	136143	1462	**53**, 80, 123, 135, 139, 445, 1025, 2745, 3127, 6129, and many more
130.85.17.45	1179206	erk177pc-1.umbc.edu.	142819	18	**80**, **135**, **139**, **445**, **1025**, **2745**, **3127**, **3410**, 5000, 6129, 443, 21
130.85.1.4	1072522	UMBC4.UMBC.EDU.	57767	606	**53**, 123, 45917, 10123, rest ephemeral
130.85.80.224	910257	pplant-80-224.pooled.umbc.edu.	142819	18	80, **135**, **139**, **445**, **1025**, **2745**, **3127**, 3410, 7000, **6129**
130.85.112.189	713587	SOA (UMBC3.UMBC.EDU.)	530123	1	**135**
130.85.81.39	694881	erk-81-39.pooled.umbc.edu.	694139	17	80, **135**, 443, 8080
130.85.111.51	553598	trc208pc-02.engr.umbc.edu.	77594	10	135, 139, 445, 1025, 1981, 2745, 3127, 3410, 6129, 7000
130.85.112.193	553284	SOA (UMBC3.UMBC.EDU.)	71631	13	80, 135, 139, 443, 445, 1025, 1981, 2745, 3127, 3410,

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

					5000, 6129, 7000
130.85.43.7	484134	SOA (UMBC3.umbc.edu.)	355638	17	80, **135**, 7000
130.85.84.186	447533	engr-84-186.pooled.umbc.edu.	60526	14	135, 139, 445, 1025, 2745, 3127, 3410, 5000, 6129

2.3.7 Top 10 Scan External Targets

Table 34 shows the top 10 systems out of 2,502,696 with a destination IP address of an external host triggering the scan detection engine.

Table 34 - Top 10 Scan External Targets

External Destination IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IPs	Unique Dest Ports	Key Dest Ports (over 10,000 in bold)
192.26.92.30	75863	c.gtld-servers.net.	3	1	**53**
128.8.10.90	73847	d.root-servers.net.	3	1	**53**
69.6.25.84	60176	SOA (ns1.wholesalebandwidth.com.)	2	1	**53**
128.63.2.53	59512	h.root-servers.net.	3	1	**53**
69.6.25.125	54086	ns1.wholesalebandwidth.com.	2	1	**53**
192.5.6.30	51127	a.gtld-servers.net.	3	1	**53**
198.41.0.4	48855	a.root-servers.net.	3	1	**53**
192.48.79.30	48309	j.gtld-servers.net.	2	1	**53**
64.21.49.27	42438	SOA (ns1.nac.net. dnsadmin.nac.net.)	2	1	**53**
203.20.52.5	41545	SOA (wombat.aussie.net. root.aussie.net.)	2	1	**53**

2.4 Scans Details

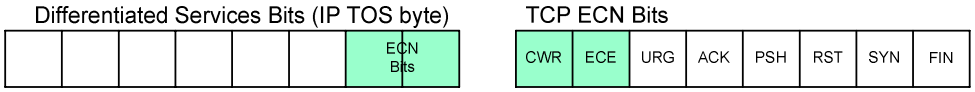
This section discusses some of the suspicious activity identified by the tables in section 2.3 Scans Summary. Through out discussions in this section I have colour coded some of the tables to give a better idea as to what systems I think are of a greater priority to investigate

Colour		Definition
Red		These systems are a priority for investigation
Orange		These systems should be investigated
Yellow		These systems are of potential concern
Green		These systems should be OK

2.4.1 Scans #1 – Suspicious flag combinations

This section will look at flag combinations trying to identify if they are legal or illegal. Some flag combinations are legal and easily explained while others are illegal and are a sure sign of mischievous behaviour. There are good reasons for attackers to use unusual flag combinations[167] [168], these include evasion of IDS, penetration of firewalls, or simply host identification, nmap[169] uses unusual flag combinations for host discovery.

The scan detection software triggers when either or both reserved bits are been set in byte 13 of the TCP header. At one point in time, it was unusual to see either of these bits set in legitimate traffic, while today it is common as they are now associated with Explicit Congestion Notification (ECN) as detailed in RFC3168. These bits work in tandem with the two low order bits in the IP headers TOS byte. Here is a quick diagram of which bits are set as detailed in the SANS course notes[170], as we don't have information on what bits where set in the IP header we can't be 100% sure that these flags are correctly set, Table 35 highlights some of the flag settings that are consistent with expected ECN traffic.



Notes	TOS 1	TOS 0	CWR bit	ECE bit	Addition Notes
SYN	0	0	1	1	During the three way hand shake of ECN-aware end-hosts
SYN-ACK	0	0	0	1	
ACK	0	1	0	0	

[167] "ambiguities in implementations of the TCP/IP" http://www.securityfocus.com/archive/1/296122/2002-10-19/2002-10-25/2
[168] Article on how TCP/IP implementations handle unusual flag combinations inconsistently http://www.kb.cert.org/vuls/id/464113
[169] nmap fingerprinting techniques are described in this classic text – http://www.insecure.org/nmap/nmap-fingerprinting-article.html
[170] Track 3 – Intrusion Detection In-Depth course notes (SANS Darling Harbour 2004) book 3.2,3.3 page 5-21

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

No congestion	0	1	0	0	(or) ECN-Aware is 01 or 10 in
No congestion	1	0	0	0	Differentiated Services Byte
Congestion	1	1	0	0	
Receiver Response	1	1	0	1	
Sender Response	1	1	1	1	

Table 35 – Scans Alerts that could be Related to ECN Type Traffic

Type	Flags	#	Comments
SYN	12****S*	7059	First stage of an ECN aware three way hand shake
UNKNOWN	*2*A**S*	489	Second stage of an ECN aware three way hand shake
UNKNOWN	*2***R**	40	Abrupt close of an ECN aware session as a receiver response
UNKNOWN	12***R**	33	Abrupt close of an ECN aware session as a sender response

Table 36 shows some of the more suspicious flag combinations that triggered the scan detection engine, hosts that generated these flag combinations need further investigation.

Table 36 – Selection of Scan Alerts with Suspicious Flag Combinations

Type	Flags	#	Comments
NULL	********	913	At a minimum ACK should be set
INVALIDACK	***A*R*F	432	RST and FIN perform different close functions
VECNA[171]	****P***	150	There is no ACK set
NOACK	**U*P*S*	135	Standard 3-way handshake no other flags except a SYN should be set
NOACK	**U**RSF	113	SYN+FIN=illegal
NOACK	**U**RS*	109	SYN+RST=illegal
XMAS[172]	**U*P**F	30	Unusual flag combination
FULLXMAS	12UAPRSF	18	Completely illegal
NOACK	1****RSF	14	SYN+RST+FIN=totally illegal
NOACK	*2U**RSF	12	Completely illegal

2.4.2 Scans #2 – Reflexive Ports

As discussed in 2.2.4 with a few exceptions packets with the same source and destination ports are unexpected behaviour. This section will expand on our previous discussion, as there are a different set of suspicious ports within the scan data.

Port 123: The NTP traffic previously seen was diffidently suspicious triggering a specific snort rule, those hosts did not appear as sources in the scan data.

```
awk -F: '{print $5":"$7":"$4":"$8}' scans.clean.csv | grep "^123:123:" | cut -d":" -f 3,4 |
sort | uniq -c | sort -rn
awk -F: '{print $5":"$7":"$6":"$8}' scans.clean.csv | grep "^123:123:" | cut -d":" -f 3,4 |
sort | uniq -c | sort -rn
```

Src IP (3)		#	FQDN
130.85.1.3		10420	umbc3.umbc.edu.
130.85.1.4		8449	UMBC4.UMBC.EDU.
209.164.32.205		1	209.164.32.205.ptr.us.xo.net.

Dst IP (53)		#	FQDN
192.5.5.250		2566	clock.isc.org.
169.254.25.129		2475	SOA (prisoner.iana.org.)
65.107.99.68		1722	65.107.99.68.ptr.us.xo.net.
204.91.240.100		1215	lithium.elemental.org.
192.5.41.40		1162	ntp0.usno.navy.mil.
128.4.1.2		1112	mizbeaver.udel.edu.
129.6.15.29		1107	time-b.nist.gov.
128.118.46.3		1035	zinc.ops.tns.its.psu.edu.
66.80.148.142		564	ip-66-80-148-142.iad.megapath.net.
66.93.55.96		378	dsl093-055-096.blt1.dsl.speakeasy.net.

All the scans are "`UDP`" except one from an external source showing "`VECNA **U*P***`" this host did perform a lot of scan activity targeting several systems, a more detailed look at this activity is in section 3.1

"`umbc3 umbc.edu`" and "`umbc4 umbc.edu`" appear to provide many vital services for the University including DNS and time synchronisation. From this traffic it looks like these systems are synchronising their time with multiple time servers around the world, making this traffic likely to be server to server which does use port 123 as its source and destination as described in section 2.2.4. What is not normal is to configure a single system to synchronise with more than a couple of external time servers, this could indicate

[171] Information on VECNA scan types http://www.securityfocus.com/archive/1/42136
[172] Further Information on XMAS tree scans http://archives.neohapsis.com/archives/snort/2002-02/0152.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

these systems are a gateway for the internal network, being a University each system is likely to be configured to synchronise with different systems.

The systems in orange look a little odd as the PTR records indicate they are DSL or dial up accounts, while the system in red is an APIPA address, as discussed in 2.2.1 these addresses are automatically configured on windows hosts. It should be verified that these addresses are correct to be leaving these systems.

Port 4672: The "Remote file access server" is assigned this port, with the observed traffic pattern, low volume of source addresses and many destinations, I suspect this traffic is related more to P2P networking[173]. eDonkey2000 uses UDP port 4672 for clients to connect to when asking for resources.

Src IP (4)	#	FQDN
130.85.111.34	13333	fsc2.engr.umbc.edu.
130.85.97.86	210	ppp-086.dialup.umbc.edu.
130.85.153.81	13	refweb17.libpub.umbc.edu.
82.83.43.1	1	dsl-082-083-043-001.arcor-ip.net.

Dst IP (9169)	#	FQDN
221.169.128.11	40	221-169-128-11.adsl.static.seed.net.tw.
140.113.239.67	40	E067.Life.NCTU.edu.tw.
62.243.219.154	35	cpe.atm2-0-1081008.0x3ef3db9a.albnxx11.customer.tele.dk.
80.32.141.236	30	236.Red-80-32-141.pooles.rima-tde.net.
212.191.160.138	29	pc138.smrw.lodz.pl.
140.121.125.129	29	yljeng4.me.ntou.edu.tw.
213.41.156.164	28	nydglc.net1.nerim.net.
213.245.92.48	28	chp-gw-04-213245092048.chello.fr.
212.179.104.88	27	cablep-179-104-88.cablep.bezeqint.net.
81.34.231.11	26	11.Red-81-34-231.pooles.rima-tde.net.

All but the external source address alerted on UDP, `82.83.43.1` came up with a scan alert "`INVALIDACK 1**A*RS* RESERVEDBITS`" which had a destination of `130.85.82.109` this host performed a lot of scan activity directed to this one host, a more detailed look at this activity is in section 3.2

Port 0: This port was discussed previously in section 2.2.4 with that conclusion being relevant here as well, again all traffic is from external systems. The systems in bold where also seen in section 2.2.4

Src IP (169)	#		FQDN
66.238.42.230	280	UDP	66.238.42.230.ptr.us.xo.net.
209.164.32.205	199	NULL ********	209.164.32.205.ptr.us.xo.net.
82.83.43.1	120	NULL ********	dsl-082-083-043-001.arcor-ip.net.
61.48.8.56	108	NULL ********	SOA (ns1.apnic.net.)
209.164.32.205	39	NOACK **U**RS*	209.164.32.205.ptr.us.xo.net.
210.130.219.28	37	UDP	SOA (ns00.cdn-japan.com.)
83.28.245.202	32	NULL ********	bmz202.neoplus.adsl.tpnet.pl.
209.164.32.205	28	NOACK **U**RSF	209.164.32.205.ptr.us.xo.net.
63.250.197.48	23	UDP	wmcontent80.bcst.yahoo.com.
68.218.142.172	20	NULL ********	adsl-218-142-172.jax.bellsouth.net.

Dst IP (150)	#		FQDN
130.85.18.25	280	UDP	shsfrontpc-02.umbc.edu.
130.85.82.109	120	NULL ********	oit-82-109.pooled.umbc.edu.
130.85.112.209	108	NULL ********	SOA)UMBC3.umbc.edu.)
130.85.97.43	84	NULL ********	ppp-043.dialup.umbc.edu.
130.85.97.55	77	NULL ********	ppp-043.dialup.umbc.edu.
130.85.12.6	46	NULL ********	mxin.umbc.edu.
130.85.53.111	37	UDP	ecs021pc34.ucslab.umbc.edu.
130.85.153.97	33	NULL ********	refweb33.libpub.umbc.edu.
130.85.70.72	32	NULL ********	whoopazz.ucs.umbc.edu.
130.85.81.116	30	NULL ********	erk-81-116.pooled.umbc.edu.

Port 6112: This port is associated with the "CDE subprocess control", "FSGS (game)[174]" and "dtspcd" all the alerts generated triggered as "`UDP`" scan.

Src IP (7)	#	FQDN
130.85.97.40	584	ppp-040.dialup.umbc.edu.
130.85.97.18	120	ppp-018.dialup.umbc.edu.
130.85.97.34	103	ppp-034.dialup.umbc.edu.
130.85.97.54	27	ppp-054.dialup.umbc.edu.
130.85.97.35	23	ppp-035.dialup.umbc.edu.
130.85.97.27	13	ppp-027.dialup.umbc.edu.
66.238.42.230	1	66.238.42.230.ptr.us.xo.net.

Dst IP (304)	#	FQDN
69.165.77.188	106	69-165-77-188.sbtnvt.adelphia.net.
65.50.129.71	106	CPE0004758421e2-CM.cpe.net.cable.rogers.com.
24.174.97.119	105	cs2417497-119.houston.rr.com.
24.107.231.84	105	cpe-24-107-231-84.ma.charter.com.
67.69.156.23	69	Toronto-HSE-ppp3828528.sympatico.ca.
67.39.183.198	10	ppp-67-39-183-198.dsl.emhril.ameritech.net.
68.223.6.228	7	adsl-223-6-228.aep.bellsouth.net.
209.91.129.140	7	sud-tcs1-port120.vianet.on.ca.
172.165.231.143	7	ACA5E78F.ipt.aol.com.
66.118.100.199	6	100-199-dial.xtn.net.

The odd external source targeted `130.85.18.25` over many ports, it is looked at with more detail in section 3.3 while the rest of the source addresses appear to come from the same network range `130.85.97.x` assigned to dial up users, this segment is quite active when

[173] http://mldonkey.berlios.de/modules.php?name=Wiki&pagename=WhatFirewallPortsToOpen
[174] quick description of FSGS http://www.ausstarcraft.com/fsgs.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

we look back at Table 9. If this is game traffic, the users behind these six systems could all be playing together or the same person dialled up at different times.

Port 6346: This port is associated with P2P network applications like gnutella [175] (BearShare, limewire, etc. [176])". The traffic pattern of a few internal hosts targeting many external systems are typical of this activity, this port number is the default for gnutella.

Src IP (11)	#	FQDN
130.85.43.2	153	SOA (UMBC3.umbc.edu.)
130.85.97.22	131	ppp-022.dialup.umbc.edu.
130.85.97.78	88	ppp-078.dialup.umbc.edu.
130.85.97.27	84	ppp-027.dialup.umbc.edu.
130.85.82.81	48	oit-82-81.pooled.umbc.edu.
130.85.98.56	21	resppp-56.dialup.umbc.edu.
130.85.97.105	9	ppp-105.dialup.umbc.edu.
130.85.112.204	8	SOA (UMBC3.umbc.edu.)
130.85.97.183	3	ppp-183.dialup.umbc.edu.
130.85.43.3	3	SOA (UMBC3.umbc.edu.)
130.85.97.100	2	ppp-100.dialup.umbc.edu.

Dst IP (520)	#	FQDN
69.244.44.120	3	pcp09209576pcs.pimaco01.az.comcast.net.
67.162.100.232	3	c-67-162-100-232.client.comcast.net.
24.51.236.159	3	fl-nworge-cmts3a-159.orlnfl.adelphia.net.
82.121.75.244	2	AAubervilliers-152-1-13-244.w82-121.abo.wanadoo.fr.
81.101.122.77	2	cpc4-ruth1-4-0-cust77.renf.cable.ntl.com.
69.198.14.152	2	CPE000039de8d60-CM000039de8c60.cpe.net.cable.rogers.com.
69.162.235.86	2	69-162-235-86.ironoh.adelphia.net.
68.74.177.12	2	adsl-68-74-177-12.dsl.emhril.ameritech.net.
68.7.184.179	2	ip68-7-184-179.sd.sd.cox.net.
68.62.226.45	2	pcp01538250pcs.huntsv01.al.comcast.net.

Port 2745: All the traffic is from internal to external systems and could be associated with the "Bagle Virus Backdoor" (Beagle.C through Beagle.K) or Urbis geolocation service, which is apparently no longer operational[177]. All the internal systems that triggered this scan alert are in bold under Scans #6 – Other Scan Ports. These systems are suspicious and should be inspected for possible infection by this virus.

Port 3127: The "W32/MyDoom" and "W32.Novarg.A backdoor" reside on this port making the internal source addresses rather suspicious. The systems triggering these alerts are listed in bold within Scans #6 – Other Scan Ports. A recent worm "W32.Mockbot.A.Worm" connects this port with 3410[178], section 2.4.6 shows this correlation.

Port 3531: This traffic triggered the "`UDP`" scan detection. PeerEnabler part of the KaZaA Media Desktop uses this port[179], this traffic is associated with P2P networking. The University should inspect these systems and remove the software if it is against policy[180].

Port 5000: All the internal systems that triggered this scan alert are in bold under Scans #6 – Other Scan Ports. There are two types of alerts triggered by this traffic, "`UDP`" and "`SYN ******S*`" which may indicate two different causes. The "`UDP`" scans could be related to "Yahoo Voice Chat"[181] while the "`TCP`" connections could be related to a worm, "Kibuv.b" was identified less than a month after these logs where recorded[182], the orange systems should be investigated for worm activity.

Port 3531 Internal Source			
Src IP (11)		#	FQDN
130.85.69.254		26	lib-69-254.pooled.umbc.edu.
130.85.69.214		25	lib-69-214.pooled.umbc.edu.
130.85.80.119		16	ss-80-119.pooled.umbc.edu.
130.85.153.33		16	refweb04.libpub.umbc.edu.
130.85.69.232		10	lib-69-232.pooled.umbc.edu.
130.85.153.31		6	refweb02.libpub.umbc.edu.
130.85.97.76		5	ppp-076.dialup.umbc.edu.
130.85.97.13		4	ppp-013.dialup.umbc.edu.
130.85.97.85		3	ppp-085.dialup.umbc.edu.
130.85.97.30		3	ppp-030.dialup.umbc.edu.
130.85.97.177		1	ppp-177.dialup.umbc.edu.

Port 5000 Internal Source				
Src IP (5)		#	Scan Type	FQDN
130.85.97.248		49	UDP	ppp-248.dialup.umbc.edu.
130.85.97.12		16	UDP	ppp-012.dialup.umbc.edu.
130.85.112.193		5	SYN ******S*	SOA (UMBC3.UMBC.EDU.)
130.85.80.119		2	SYN ******S*	ss-80-119.pooled.umbc.edu.
130.85.153.195		2	SYN ******S*	SOA (UMBC3.umbc.edu.)

[175] http://mldonkey.berlios.de/modules.php?name=Wiki&pagename=WhatFirewallPortsToOpen
[176] http://www.torontotechcenter.com/ports6001_7000.shtml
[177] http://www.chebucto.ns.ca/~rakerman/port-table.html
[178] http://securityresponse.symantec.com/avcenter/venc/data/w32.mockbot.a.worm.html
[179] http://www.derkeiler.com/Mailing-Lists/securityfocus/incidents/2003-07/0043.html
[180] The next message in the thread describes the process to identify if it is KaZaA or not http://www.derkeiler.com/Mailing-Lists/securityfocus/incidents/2003-07/0055.html
[181] http://isc.sans.org/show_comment.php?id=759
[182] http://www.internetweek.com/shared/printableArticle.jhtml?articleID=20301309

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Port 3410: Most of the destination systems are in the `130.x.x.x` range, this traffic triggered as "`SYN ******S*`". All the internal systems that triggered this scan are seen in Scans #6 – Other Scan Ports under port 3410 in bold.

2.4.3 Scans #3 – Port 53 (DNS)

Most of the DNS traffic logged in the scan files I would consider benign.

Port	Count	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
53	4681210	Domain Name Server	13	66	14	6	115665

There are 72 systems triggering this alert, the top two are internal systems making up 99% of the total traffic.

Source IP address		Count	FQDN (Fully Qualified Domain Name)	Comments
130.85.1.3		3617968	UMBC3.UMBC.EDU.	Appears to be a key UMBC DNS server
130.85.1.4		1063009	umbc4.umbc.edu.	Appears to be a key UMBC DNS server
130.85.27.3		36	vodka.umbc.edu.	Maybe configured to perform its own queries
130.85.97.82		9	ppp-082.dialup.umbc.edu.	manually configured or scanning for DNS
130.85.111.34		5	fsc2.engr.umbc.edu.	manually configured or scanning for DNS
130.85.69.232		1	lib-69-232.pooled.umbc.edu.	manually configured or scanning for DNS

The top 10 targets from these addresses are the same as detailed in Table 34 with at least six being top-level name servers[183], this traffic appears to be normal DNS traffic occurring from the internal network. The orange and yellow systems may need some further investigation as they could be scanning if these hosts do not expect DNS traffic. Traffic from external IP address are targeting more than just the University DNS servers. The systems marked with red require investigation for scan activity.

Source IP address		#	FQDN (Fully Qualified Domain Name)	Comments
130.85.1.3		57	UMBC3.UMBC.EDU.	Appears to be a key UMBC DNS server
130.85.1.5		35	umbc5.umbc.edu.	Appears to be a key UMBC DNS server
130.85.1.4		30	umbc4.umbc.edu.	Appears to be a key UMBC DNS server
130.85.18.25		16	shsfrontpc-02.umbc.edu.	The source of this traffic should be investigated
130.85.97.55		11	ppp-055.dialup.umbc.edu.	The source of this traffic should be investigated
130.85.97.43		9	ppp-043.dialup.umbc.edu.	The source of this traffic should be investigated
130.85.82.109		7	oit-82-109.pooled.umbc.edu.	The source of this traffic should be investigated
130.85.81.116		6	erk-81-116.pooled.umbc.edu.	The source of this traffic should be investigated
130.85.112.209		6	-	The source of this traffic should be investigated
130.85.97.73		1	ppp-073.dialup.umbc.edu.	The source of this traffic should be investigated
130.85.97.65		1	ppp-065.dialup.umbc.edu.	The source of this traffic should be investigated
130.85.53.50		1	ecs021pc20.ucslab.umbc.edu.	The source of this traffic should be investigated
130.85.34.14		1	imap.cs.UMBC.EDU.	The source of this traffic should be investigated
130.85.11.4		1	quarantine.UMBC.EDU.	The source of this traffic should be investigated

Recommendations

I would consider these as false positives, they appear to trigger on normal DNS behaviour, I expect `130.85.1.3` and `130.85.1.4` are primary name servers performing iterative lookups of domain names for clients internal to the network. Include these and any other known DNS servers within the portscan-ignore hosts variable to reduce the false positives.

2.4.4 Scans #4 – Windows Ports (135,139, and 445)

There is a lot of scan activity related to these ports, which are all in some way related to windows based traffic.[184] These ports are constantly scanned for vulnerabilities[185] [186] [187] and have a prominent place in the top 10 target ports on dshield[188]

[183] http://pigtail.net/LRP/gtld-servers.html
[184] http://www.microsoft.com/resources/documentation/windows/2000/server/reskit/en-us/tcpip/part4/tcpappc.mspx
[185] Port 135 – http://www.dshield.org/port_report.php?port=135&recax=1&tarax=2&srcax=2&percent=N&days=70&Redraw=
[186] Port 139 – http://www.dshield.org/port_report.php?port=139&recax=1&tarax=2&srcax=2&percent=N&days=70&Redraw=
[187] Port 445 – http://www.dshield.org/port_report.php?port=445&recax=1&tarax=2&srcax=2&percent=N&days=70&Redraw=
[188] http://www.dshield.org/topports.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Port	Count	Known traffic using this port number	Unique Alerts	Unique External Src IP	Unique Internal Dest IP	Unique Internal Src IP	Unique External Dest IP
135	2616043	DCE endpoint resolution, NCS local location broker	8	124	263	20	1844430
445	582272	Win2k+ Server Message Block	4	21	259	15	428583
139	473811	NETBIOS Session Service, [trojan] SMB Relay, [trojan] Sadmind	6	30	259	19	346566

Table 37 shows, which internal systems generated scan activity on the selected ports and the quantity of target addresses across the whole scan data:

```
awk -F: '{print $7":"$4}' scans.clean.csv | grep "^[port]:" | cut -d: -f 2 | grep
"^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | so.rt -rn
awk -F: '{print $4":"$6}' scans.clean.csv | grep "^[ipaddr]:" | cut -d: -f 2 | sort | uniq |
wc -l
```

Table 37 – Internal hosts scanning Windows Ports Scanned

IP address		# to 135	# to 139	# to 445	FQDN	Target IP covering any port
130.85.17.45		164238	132575	144934	erk177pc-1.umbc.edu.	142819
130.85.43.4		-	6	-	-	1091
130.85.43.7		480932	-	-	-	355638
130.85.43.8		8467	4587	6623	-	15059
130.85.43.10		94991	1563	48259	-	119007
130.85.43.13		924	571	752	-	1516
130.85.69.155		891	666	801	lib-69-155.pooled.umbc.edu.	1030
130.85.69.210		13314	12108	12835	lib-69-210.pooled.umbc.edu.	17017
130.85.70.164		-	3	-	dojo.ucs.umbc.edu.	1286
130.85.80.71		1335	741	1035	ss-80-71.pooled.umbc.edu.	1128
130.85.80.119		7809	7666	7779	ss-80-119.pooled.umbc.edu.	11160
130.85.80.224		116671	110408	114309	pplant-80-224.pooled.umbc.edu.	100635
130.85.81.39		693980	-	-	erk-81-39.pooled.umbc.edu.	694139
130.85.109.25		23036	-	-	ecs10202.engr.umbc.edu.	17406
130.85.111.34		-	44	-	fsc2.engr.umbc.edu.	17408
130.85.111.51		81305	55848	71532	trc208pc-02.engr.umbc.edu.	77594
130.85.112.189		713587	-	-	-	530123
130.85.112.193		68644	55966	63222	-	71631
130.85.112.225		-	2	-	-	1384
130.85.150.210		9929	6466	8519	libpc10.lib.umbc.edu.	7923
130.85.150.226		23579	16933	20787	libpc22.lib.umbc.edu.	19903
130.85.153.195		29934	24497	27585	-	27905

All these systems are targeting more than a thousand different external IP address, this is a sure sign of some scan activity potentially associated with a worm. All these systems should be checked for Trojans and worms.

Recommendations

These ports should be blocked form entering or leaving the University network, the University of California, Irvine did just this[189] providing strong reasons why these ports should not be accessible from the Internet.

2.4.5 Scans #5 – Web based scans (Port 80 and 443)

These two ports are synonymous with web traffic, port 80 is used mainly for HTTP traffic without encryption while port 443 is set aside for encrypted HTTP traffic. There are numerous issues with these protocols and scanning for these ports is continuous as can be see on the dshield website[190] [191], these figures show there was heightened interest in these ports during the time frame these logs where captured, at the time of writing port 80 had a place in the top 10 target ports on dshield[192].

[189] http://www.nacs.uci.edu/security/netbios.html
[190] Port 80 – http://www.dshield.org/port_report.php?port=80&recax=1&tarax=2&srcax=2&percent=N&days=70&Redraw=
[191] Port 443 – http://www.dshield.org/port_report.php?port=443&recax=1&tarax=2&srcax=2&percent=N&days=70&Redraw=
[192] http://www.dshield.org/topports.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

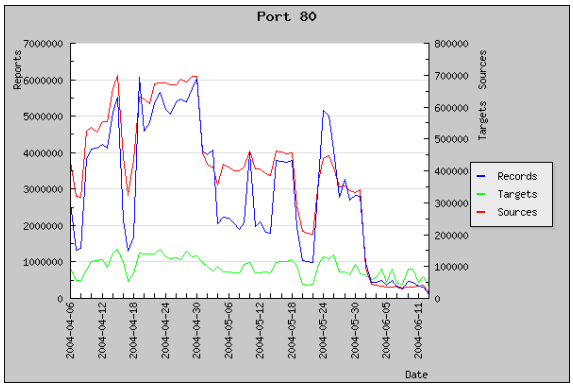


Figure 22 - DShield.org port 80 report (70 day trend)

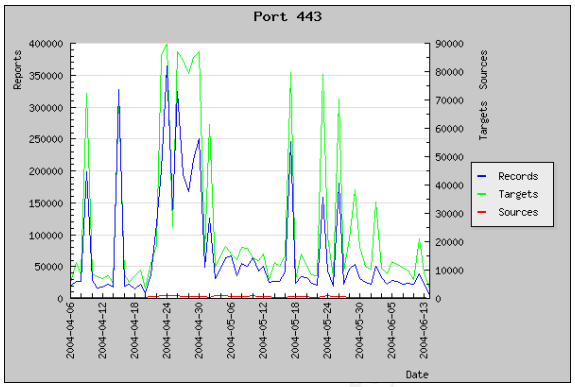


Figure 23 - DShield.org port 443 report (70 day trend)

There are vulnerabilities with various web servers and numerous worms created to take advantage of these flaws like code red[193] [194], Nimda[195] [196], and WebDav Buffer Overflow Attacks[197] [198]. A lot of this scan activity could be related to automatic scanning performed by various worms.

Port	Count	Known traffic using this port number	Unique Alerts	Unique Ext Src IP	Unique Int Dest IP	Unique Int Src IP	Unique Ext Dest IP
80	604263	World Wide Web HTTP	21	255	15496	146	320430
443	297002	HTTP protocol over TLS SSL	11	42	15725	29	33

The figures for port 80 look suspiciously like scan activity, a low internal source count with a high level of destinations, for internal hosts the port 443 traffic looks OK. As far as external traffic is concerned, there is that classic scan pattern. This indicates there is some activity on port 445 possibly a worm that is scanning for victims, at this stage internal systems do not appear to be affected. This table shows the top 10 unique alerts that triggered the scan detection software.

Port 80 scan		Count
SYN ******S*		603647
UDP		1058
SYN 12****S* RESERVEDBITS		924
VECNA ****P***		44
FIN *******F		26
NOACK **U**RSF		8
NULL ********		7
NOACK **U**RS*		6
NOACK **U*P*S*		5
VECNA **U*P***		2

Port 443 scan		Count
SYN ******S*		297076
UDP		73
SYN 12****S* RESERVEDBITS		17
NOACK **U*P*S*		2
NOACK **U**RS*		2
VECNA ****P**F		1
UNKNOWN 12*A**S* RESERVEDBITS		1
NOACK 1***PRSF RESERVEDBITS		1
NOACK 1****RSF RESERVEDBITS		1
NOACK **U*PR*F		1

I would not be inclined to investigate the SYN alerts based on this table, these alerts are very generic and like the SYN alerts with the reserved bits set (see section 2.4.1), they look legitimate. Signs of scan activity by hosts will become more apparent when looking at the top talker tables. Alerts in red are diffidently illegal, the hosts generating these packets should be crossed referenced with other traffic looking for signs of scan activity. The orange alerts are unusual and worth further investigation.

Recommendations

The alert data is more valuable when trying to determine issues with web traffic, the reason being signatures looking for specific illegal traffic that is more readily recognisable by the analyst like the "`NIMDA - Attempt to execute cmd from campus host`" signature used by the University. IP addresses triggering these alerts should be cross-referenced

[193] http://www.cert.org/advisories/CA-2001-19.html
[194] http://www.linklogger.com/coderedii.htm
[195] http://www.cert.org/advisories/CA-2001-26.html
[196] http://www.linklogger.com/nimda.htm
[197] http://www.cert.org/advisories/CA-2003-09.html
[198] http://www.linklogger.com/webdav.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

with the scan data to paint a more accurate picture of the hosts activity. The scans with unusual flag combinations need further investigation as they are likely to be associated with no good.

2.4.6 Scans #6 – Other Scan Ports

If you ever question why we see so much scan activity, the answer is because it works, attackers search the internet for open services to compromise or listening Trojans in the hope of finding systems already compromised. An article titled "How thorough is Trojan detection by port scanning?" discusses from the administrators point of view how valuable it is to scan the network for Trojan ports, which is certainly relevant for attackers alike. http://www.diamondcs.com.au/index.php?page=archive&id=sec05

Evidence of Scanning from Internal Systems

Port 3127, 3410 and 6129: The "W32/MyDoom" and "W32.Novarg.A backdoor" are associated with port 3127 though it may choose any port from 3127 to 3198 this is the most common, at the time of writing, this port is one of the top 10 targets on dshield[199]. Systems in bold produced alerts that had both the source and destination port of 3127.

Table 38 - Port 3127 Internal Sources

IP address	Count	IP address	Count	IP address	Count	IP address	Count
130.85.17.45	140536	130.85.69.226	22	130.85.97.17	128	**130.85.112.193**	60799
130.85.43.8	5070	130.85.69.232	59	130.85.97.171	771	130.85.150.210	7651
130.85.43.10	3591	**130.85.80.119**	7660	130.85.97.22	2	130.85.150.226	19221
130.85.43.13	724	**130.85.80.224**	112797	130.85.97.38	2	**130.85.153.195**	26286
130.85.69.155	755	130.85.80.71	928	130.85.97.84	2		
130.85.69.210	12686	**130.85.84.186**	48218	130.85.110.72	6		
130.85.69.214	101	130.85.97.115	153	**130.85.111.51**	66311		

A more recent worm "W32.Mockbot.A.Worm" links this port with 3410[200] and 6129, the IP addresses in blue generate traffic with two or more of these ports giving a strong case to suggest these systems are compromised by this worm. It propagates by looking for systems previously infected by "W32.Mydoom.A@mm" or "Backdoor.Optix" worms.

Table 39 - Port 3410 Internal Sources

IP address	Count	IP address	Count	IP address	Count	IP address	Count
130.85.17.45	24332	130.85.69.155	650	**130.85.80.224**	95063	130.85.97.92	1
130.85.43.8	4186	**130.85.69.210**	12181	**130.85.84.186**	39471	130.85.110.72	2
130.85.43.10	1258	130.85.69.214	115	130.85.97.115	66	**130.85.111.51**	52700
130.85.43.13	532	130.85.69.226	8	130.85.97.17	80	**130.85.112.193**	54513
130.85.53.225	21	130.85.69.232	97	130.85.97.171	293	**130.85.150.226**	15855
130.85.53.41	2	**130.85.80.119**	7690	130.85.97.39	2	**130.85.153.195**	7535

Table 40 - Port 6129 Internal Sources

IP address	Count	IP address	Count	IP address	Count	IP address	Count
130.85.17.45	136106	130.85.69.210	12479	130.85.97.115	91	130.85.150.210	6965
130.85.43.8	4808	130.85.80.71	845	130.85.97.171	613	130.85.150.226	18002
130.85.43.10	2163	130.85.80.119	7728	130.85.110.72	2	130.85.153.195	25519
130.85.43.13	666	130.85.80.224	112015	130.85.111.34	2		
130.85.53.225	1	130.85.84.186	44987	130.85.111.51	62093		
130.85.69.155	705	130.85.97.17	117	130.85.112.193	58833		

At a minimum systems marked with red should be inspected for viruses, orange systems are suspicious, while yellow systems could be a false positive as there was only one or two counts.

Port 2745: Beagle.C through to Beagle.K is known to use this port, these are the internal systems that may require further investigation, the bold systems in Table 41 where also seen in Scans #2 – Reflexive Ports. Looking at the IP address that triggered the scan data we find a strong correlation with the discussion on ports 3127, 3410, 6129. Though the link to Symantec does not mention port 2745 in the description of "W32.Mockbot.A.Worm"

[199] http://www.dshield.org/topports.php
[200] http://securityresponse.symantec.com/avcenter/venc/data/w32.mockbot.a.worm.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

I think these systems are compromised by this worm or a variant that also scans for systems compromised by "Beagle" worms and its variants.

Table 41 - Port 2745 Internal Sources

IP address	Count
130.85.17.45	159058
130.85.43.8	12752
130.85.43.10	139748
130.85.43.13	1087
130.85.53.225	1
130.85.69.155	957
130.85.69.210	13598

IP address	Count
130.85.69.214	35
130.85.69.226	10
130.85.69.232	42
130.85.80.71	1611
130.85.80.119	7875
130.85.80.224	118118
130.85.84.186	66464

IP address	Count
130.85.97.13	1
130.85.97.17	395
130.85.97.82	1
130.85.97.115	507
130.85.97.171	2755
130.85.109.25	23004
130.85.110.72	3

IP address	Count
130.85.111.51	86470
130.85.112.193	72193
130.85.150.210	10868
130.85.150.226	25197
130.85.153.195	31387

Port 1025: Microsoft Remote Procedure Call (RPC) service listens on this port along with a few Trojans like Fraggle Rock, Remote Storm, and NetSpy. According to a MS KB article, it could be used as an Active Directory logon and directory replication interface[201]. Figure 24 taken from the dshield web site shows that around the same time these logs where generated there is a peak in activity. This activity looks unusual, **Error! Not a valid bookmark self-reference.** shows the internal hosts that triggered the scan detection, these systems should be investigated for possible worm activity[202].

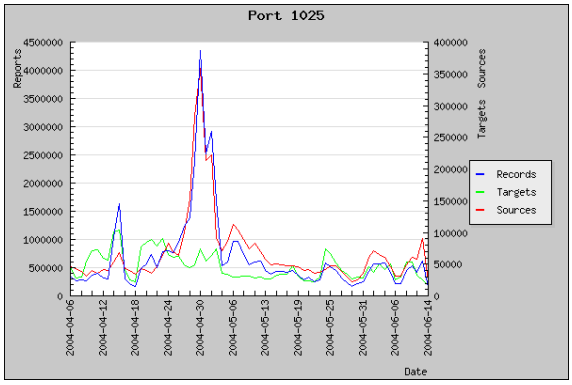


Figure 24 - DShield.org port 1025 report (70 day trend)

Table 42 - Port 1025 Internal Sources

IP address	Count
130.85.17.45	149325
130.85.53.225	12
130.85.53.41	5
130.85.69.155	851
130.85.69.210	13032
130.85.69.214	7
130.85.69.226	1

IP address	Count
130.85.69.232	3
130.85.70.164	1
130.85.70.197	2
130.85.80.119	7761
130.85.80.224	115244
130.85.80.71	1178
130.85.82.8	1

IP address	Count
130.85.83.91	68
130.85.84.186	56658
130.85.97.54	1
130.85.97.86	2
130.85.98.56	1
130.85.109.25	23032
130.85.110.72	21

IP address	Count
130.85.111.34	40
130.85.111.51	76556
130.85.112.193	66104
130.85.112.225	6
130.85.150.210	9260
130.85.150.226	22308
130.85.153.195	28660

Port 5000: Refer to section 2.4.2 for a discussion on this port. The systems in bold are also listed there.

IP address	Scan Type	Count
130.85.1.3	UDP	1
130.85.17.45	SYN ******S*	24075
130.85.43.13	SYN ******S*	474
130.85.53.225	UDP	37
130.85.69.155	SYN ******S*	7
130.85.69.210	SYN ******S*	11903
130.85.70.164	SYN ******S*	3

IP address	Scan Type	Count
130.85.80.119	SYN ******S*	7770
130.85.84.186	SYN ******S*	37030
130.85.84.235	SYN ******S*	2
130.85.97.115	SYN ******S*	39
130.85.97.12	UDP	17
130.85.97.17	SYN ******S*	83

IP address	Scan Type	Count
130.85.97.171	SYN ******S*	256
130.85.97.248	UDP	49
130.85.111.34	UDP	11
130.85.112.193	SYN ******S*	52450
130.85.153.195	SYN ******S*	7356
130.85.153.81	SYN ******S*	2

Evidence of Scanning from External Systems

Port 32773: This port is known for Sun's rpc.nisd service but does have a few worms and Trojans. The internal source and external destination figures are not that alarming, the concern is associated with 2 external sources and the 13,877 internal destinations. This looks like a clear case of network scanning across portions of the University network, these systems need further investigation and potentially reported.

Ext src addr	FQDN	#
213.189.229.5		18527
211.136.159.206		7293

[201] http://support.microsoft.com/default.aspx?scid=KB;en-us;q280132
[202] http://www.linklogger.com/TCP1025.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Taking the leading IP address `213.189.229.5` as an example we can quickly determine the sort of information that could lead to a phone call for the responsible company/ISP

Destination ports	#	Total destination IP
21	9122	11027
23	9055	
111	203	

Port 21: Not only is this the well-known port for FTP control, it does have a few worms and Trojans. As in the previous case, the internal source and external destination figures are not that alarming, on the face of it they look normal, the unusual numbers are associated with the external sources totalling 21 and the 13,527 internal destinations. I would not expect any network to have this many legitimate FTP servers, thus indicating one or more of these external IP address performed a scan across portions of the University network, these systems should be further investigated and potentially reported.

Ext src addr		FQDN	#
207.3.145.130		SOA (unix.worldpath.net.)	9122
82.49.56.134		host134-56.pool8249.interbusiness.it.	8683
218.188.13.228		SOA (bbdns1.on-nets.com.)	5801
213.180.193.68		proxychecker.yandex.net.	653
203.15.51.51		tester-51-2.sorbs.net.	604
62.58.50.220		dsbl.zonnet.nl.	137
163.20.86.130		domain.wles.tpc.edu.tw.	71
195.47.93.190		SOA (dns.nextra.cz.)	62
66.238.42.230		66.238.42.230.ptr.us.xo.net.	29
61.172.200.228		?	23

I would be a little less concerned with 213.180.193.68 as it is potentially associated with a proxy checking system as previously described in section 2.2.5 regarding port 53 traffic.

Port 20168: This port is possibly related to the lovegate virus and has figures that indicate external scanning, the source IP addresses should be investigated for scanning activity.

Port 4000: This port is related to a few different worms and Trojans and has figures that indicate external scanning, these are the top five source IP addresses that need further investigation.

Port 20168 External Source			
Ext src addr		**FQDN**	**#**
206.222.14.209		SOA (dns4.ee.net.)	16732
80.38.233.3		3.Red-80-38-233.pooles.rima-tde.net.	13892
64.166.194.201		adsl-64-166-194-201.dsl.lsan03.pacbell.net.	10389
62.58.50.220		dsbl.zonnet.nl.	1
203.15.51.51		tester-51-2.sorbs.net.	1

Port 4000 External Source			
Ext src addr		**FQDN**	**#**
193.120.129.82		host2.sdl.ie.	14580
208.182.190.91		host190-91.wa-bass-ms.davidson.k12tn.net.	8243
130.235.23.99		lukas.lucs.lu.se.	3392
213.96.185.200		200.Red-213-96-185.pooles.rima-tde.net.	1156
203.144.159.196		ppp-203.144.159.196.revip.asianet.co.th.	881

Recommendation

Systems scanning the University network are probably leading up to an attack against University hosts. These systems should be reported and traced back to their origin, there are a few reasons why these systems are scanning:

1. They have been compromised by a worm trying to propagate itself further;
2. It is a compromised system used as a sacrificial lamb to obtain as much information as they can about systems on the internet before launching an attack.
3. The attacker is actually on the other end of the system performing these scans preparing for a compromise of a system

The University is in a great position to play an important for role within the security community by providing information to resources like dshield. This can help the correlation effort for the greater Internet community[203] and add to the weight when trying to shut down attacker activity by joining fight back[204].

[203] http://www.dshield.org/intro.php
[204] http://www.dshield.org/fightback.php

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

File sharing ports

These systems should be investigated if it is against University policy to use P2P services that are widely used to share copyrighted material.

Port 6346: This port is used by the gnutella service

IP address	Count
130.85.43.13	14
130.85.43.14	33
130.85.43.16	27
130.85.43.2	384
130.85.43.3	59
130.85.43.4	2
130.85.43.5	2
130.85.43.6	4
130.85.43.7	1
130.85.53.128	1
130.85.53.225	17990

IP address	Count
130.85.53.41	3335
130.85.53.50	1
130.85.70.105	1
130.85.70.164	23
130.85.70.16	67
130.85.82.109	7
130.85.82.81	50
130.85.84.203	4
130.85.84.235	12
130.85.97.100	131
130.85.97.105	4470

IP address	Count
130.85.97.112	14
130.85.97.182	864
130.85.97.183	1745
130.85.97.22	2192
130.85.97.27	145
130.85.97.39	533
130.85.97.43	9
130.85.97.52	3
130.85.97.60	4
130.85.97.62	11
130.85.97.78	1720

IP address	Count
130.85.97.82	49
130.85.98.56	4311
130.85.111.34	1
130.85.112.204	93
130.85.112.209	1
130.85.112.225	1
130.85.153.81	9
130.85.153.97	2

Port 41170: This port is used by bluster a file sharing program

IP address	Count
130.85.97.39	2
130.85.97.56	11083

IP address	Count
130.85.97.73	1803
130.85.97.74	4984

IP address	Count
130.85.97.94	25
130.85.97.95	58

IP address	Count
130.85.97.96	8680
130.85.97.181	138

Recommendation

The University of Florida has implemented a solution that have all but eliminated the use of P2P networking on the campus network. This could be a long-term goal if this becomes more of a problem. http://www.wired.com/news/print/0,1294,60613,00.html

2.5 OOS log Summary

The tables in this section display the OOS data in various ways helping identify suspicious activity with some items discussed further in section 2.6

2.5.1 Top 10 OOS External Talkers

There where 259 external sources logged in the OOS files, Table 43 shows the top 10

Table 43 - Top 10 OOS External Talkers

External Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IP	Unique Dst Ports	Key Destination Ports
68.54.84.49	937	pcp01741335pcs.howard01.md.comcast.net.	1	1	110
199.184.165.136	131	xemacs.org.	2	131	2500, 2640, 4922, the next 128 ports are between 35819 and 36028 once each.
66.225.198.20	118	unknown.splashhost.net.	1	1	25
204.92.130.35	93	smtp25.svngsrgstr.com.	3	1	25
204.92.130.31	80	smtp21.svngsrgstr.com.	2	1	25
141.152.34.103	78	SOA (nsdc.ba-dsg.net.)	2	1	25
204.92.130.36	75	smtp26.svngsrgstr.com.	1	1	25
204.92.130.32	69	smtp22.svngsrgstr.com.	2	1	25
202.70.64.15	50	?	2	1	80
67.119.232.234	47	adsl-67-119-232-234.dsl.sndg02.pacbell.net.	1	1	110

2.5.2 Top 10 OOS Internal Targets

There where 50 internal targets logged in the OOS files, Table 44 shows the top 10

Table 44 - Top 10 OOS Internal Targets

Targeted Internal IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IP	Unique Dst Ports	Key Destination Ports
130.85.6.7	983		4	2	**110**, 80
130.85.12.6	821		149	1	25
130.85.24.118	128		1	128	Ephemeral (src 20)
130.85.5.67	67		7	1	8080
130.85.24.44	54		17	1	80
130.85.12.4	47		1	1	110
130.85.34.11	44		9	1	80

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

130.85.34.14	39		4	2	113, 25
130.85.97.101	36		5	1	3646
130.85.25.72	26		9	1	113

2.5.3 Top 10 OOS Internal Talkers

There where only two systems with an internal source address that are logged in the OOS files for the selected period.

Table 45 - Top 10 OOS Internal Talkers

Internal Source IP	Count	FQDN (Fully Qualified Domain Name)	Unique Dst IP	Unique Dst Ports	Key Destination Ports
130.85.12.6	5	mxin.umbc.edu.	5	5	60783, 48174, 4145, 3122, 2114
130.85.12.4	1	mail.umbc.edu.	1	1	52602

2.5.4 Top 10 OOS External Targets

From the two internal talkers there where six targets each receiving only one packet.

Table 46 - Top 10 OOS External Targets

Targeted External IP	Count	FQDN (Fully Qualified Domain Name)	Unique Src IP	Unique Dst Ports	Key Destination Ports
69.59.157.47	1		1	1	60783 (src 25)
68.55.137.118	1		1	1	52602 (src 993)
61.167.236.216	1		1	1	4145 (src 25)
211.144.32.80	1		1	1	2114 (src 25)
200.52.127.126	1		1	1	3122 (src 25)
161.58.155.247	1		1	1	48174 (src 25)

2.6 OOS log Details

This section discusses suspicious activity identified within the tables in section 2.5

2.6.1 OSS #1 – 68.54.84.49

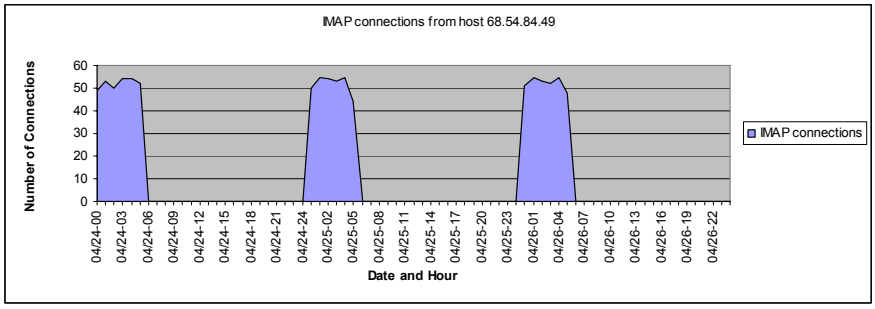
These are the first three packets out of 937 logged in the OOS log files.

```
04/24-00:05:48.635408 68.54.84.49:42379 -> 130.85.6.7:110
TCP TTL:51 TOS:0x0 ID:9047 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xF3D0181B  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1018444620 0 NOP WS: 0

04/24-00:06:54.932062 68.54.84.49:42380 -> 130.85.6.7:110
TCP TTL:51 TOS:0x0 ID:2900 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xF75C5E65  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1018451249 0 NOP WS: 0

04/24-00:07:58.248062 68.54.84.49:42381 -> 130.85.6.7:110
TCP TTL:51 TOS:0x0 ID:50510 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xFB98A8A3  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1018457581 0 NOP WS: 0
```

This traffic does not look particularly unusual, a break down of the first packet reveals:

Element	Comments
Source and destination IP	These look OK, external source, internal destination
Source and destination port	These look OK, ephemeral source port directed to IMAP port 130.85.7.110 resolves to umbc7.umbc.edu. this name does not give a clue as to the function of this machine.
TCP TTL:51	This could be a systems that has set it's TTL to 64 and is around 13 hops away.
TOS:0x0 and 12****S*	Nothing strange with the SYN set, it is the first part of a three way handshake, as previously discussed in section 2.4.1 an ECN aware host will set both reserved flags on within the initial connection, and the low order bits in the TOS byte are set to zero.
IP ID:9047	A non zero number here is good.
IpLen:20	Correct length for an IP datagram without options.
DgmLen:60	Correct length for a standard SYN datagram
DF	Don't Fragment is not so strange
Seq: 0xF3D0181B	A non zero sequence number is correct.
Ack: 0x0	A zero acknowledgement number is expected, as there is nothing for this host to acknowledge yet.
Win: 0x16D0	This is a reasonable window size (decimal 5840)
TcpLen: 40	This is expected, the datagram length is 60, the IP header is 20 so the TCP length should be 40bytes
TCP Options	These Look OK

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



The only thing really out of the ordinary with this traffic is the setting of the reserved bits. If we look at the timing of this traffic, we can surmise it is a typical IMAP client connecting to a server checking every minute for new email. Graphing the connections over time reveals a user who begins work at time `00`, connects to the IMAP server, works for six hours and disconnects only to return at the same time the next day. This traffic can be put down to a false positive.

Recommendations

The OOS detection software needs to be more aware of ECN traffic to reduce the chance of false positives.

3 External Source Address Registration

The hosts in this section have been selected as potential attackers and require further investigation, this may end in the manager of the address space being contacted to stop the activity from continuing.

3.1 209.164.32.205

This host was first identified in 2.4.2 during our discussion on port 123, there was a lot of activity from this host directing scan activity toward 12 hosts across 233 ports.

Owner		Registration Information		Contact Infromation	
OrgName:	XO Communications	**NetRange:**	209.164.0.0 - 209.164.63.255	**OrgAbuseName:**	XO Communications, Network Violations
Address:	Corporate Headquarters	**CIDR:**	209.164.0.0/18	**OrgAbusePhone:**	+1-866-285-6208
Address:	11111 Sunset Hills Road	**NetName:**	XOXO-BLK-18	**OrgAbuseEmail:**	abuse@xo.com
City:	Reston			**OrgTechName:**	XO Communications, IP Administrator
StateProv:	VA	**Comment:**	For best results, please send all spam and worm reports only to abuse@xo.com.	**OrgTechPhone:**	+1-703-547-2000
PostalCode:	20190-5339			**OrgTechEmail:**	ipadmin@eng.xo.com
Country:	US				

This address did not appear as an attacker in the DShield database

3.2 82.83.43.1

This host was first identified in 2.4.2 during our discussion of port 4672, there was a lot of activity from this host directed to `130.85.82.109` over a 3-hour period covering 161 ports, some obviously crafted like those noted in Table 36.

Owner		Registration Information		Contact Infromation	
descr:	ARCOR AG	**inetnum:**	82.82.169.0 - 82.83.255.255	**role**:	Mannesmann Arcor Network Operation Center
descr:	Alfred-Herrhausen-Allee 1	**route:**	82.82.0.0/15	**address:**	Arcor AG & Co.KG
descr:	D-65760 Eschborn	**netname:**	ARCOR-DSL-NET12	**address:**	Department TBN
country:	DE			**address:**	Otto-Volger-Str. 19
				address:	D-65843 Sulzbach/Ts.
				address:	Germany
				phone:	+49 6196 523 0864
				e-mail:	noc@adm.arcor.net
				Security issues	mailto:abuse@arcor-ip.de
				Information	http://www.arcor.net
				Peering contact	mailto:peering@adm.arcor.net
				Operational issues	mailto:noc@adm.arcor.net
				Address assignment	mailto:ip-registry@arcor.net

This address did not appear as an attacker in the DShield database

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.3 66.238.42.230

This host was identified in 2.4.2 during our discussion on port 6112, there was a lot of activity from this host towards `130.85.18.25` across 1026 UDP ports over 27 minutes.

Owner		Registration Information		Contact Infromation	
OrgName:	XO Communications	**NetRange:**	66.236.0.0 - 66.239.255.255	**OrgAbuseName:**	XO Communications, Network Violations
Address:	Corporate Headquarters	**CIDR:**	66.236.0.0/14	**OrgAbusePhone:**	+1-866-285-6208
Address:	11111 Sunset Hills Road	**NetName:**	XOX1-BLK-2	**OrgAbuseEmail:**	abuse@xo.com
City:	Reston			**OrgTechName:**	XO Communications, IP Administrator
StateProv:	VA			**OrgTechPhone:**	+1-703-547-2000
PostalCode:	20190-5339			**OrgTechEmail:**	ipadmin@eng.xo.com
Country:	US				

This address did not appear as an attacker in the DShield database

3.4 66.250.188.13

This host triggered an alert shown in section 2.2.4 under the discussion of port 123.

Owner		Registration Information		Contact Infromation	
OrgName:	Cogent Communications	**NetRange:**	66.250.0.0 - 66.250.255.255	**OrgAbuseName:**	Cogent Abuse
Address:	1015 31st Street, NW			**OrgAbusePhone:**	+1-877-875-4311
City:	Washington	**CIDR:**	66.250.0.0/16	**OrgAbuseEmail:**	abuse@cogentco.com
StateProv:	DC	**NetName:**	COGENT-NB-0001	**OrgTechName:**	Cogent Communications
PostalCode:	20007			**OrgTechPhone:**	+1-877-875-4311
Country:	US			**OrgTechEmail:**	noc@cogentco.com

This address did not appear as an attacker in the DShield database

3.5 212.204.214.70

This host triggered an alert shown in section 2.2.4 under the discussion of port 123.

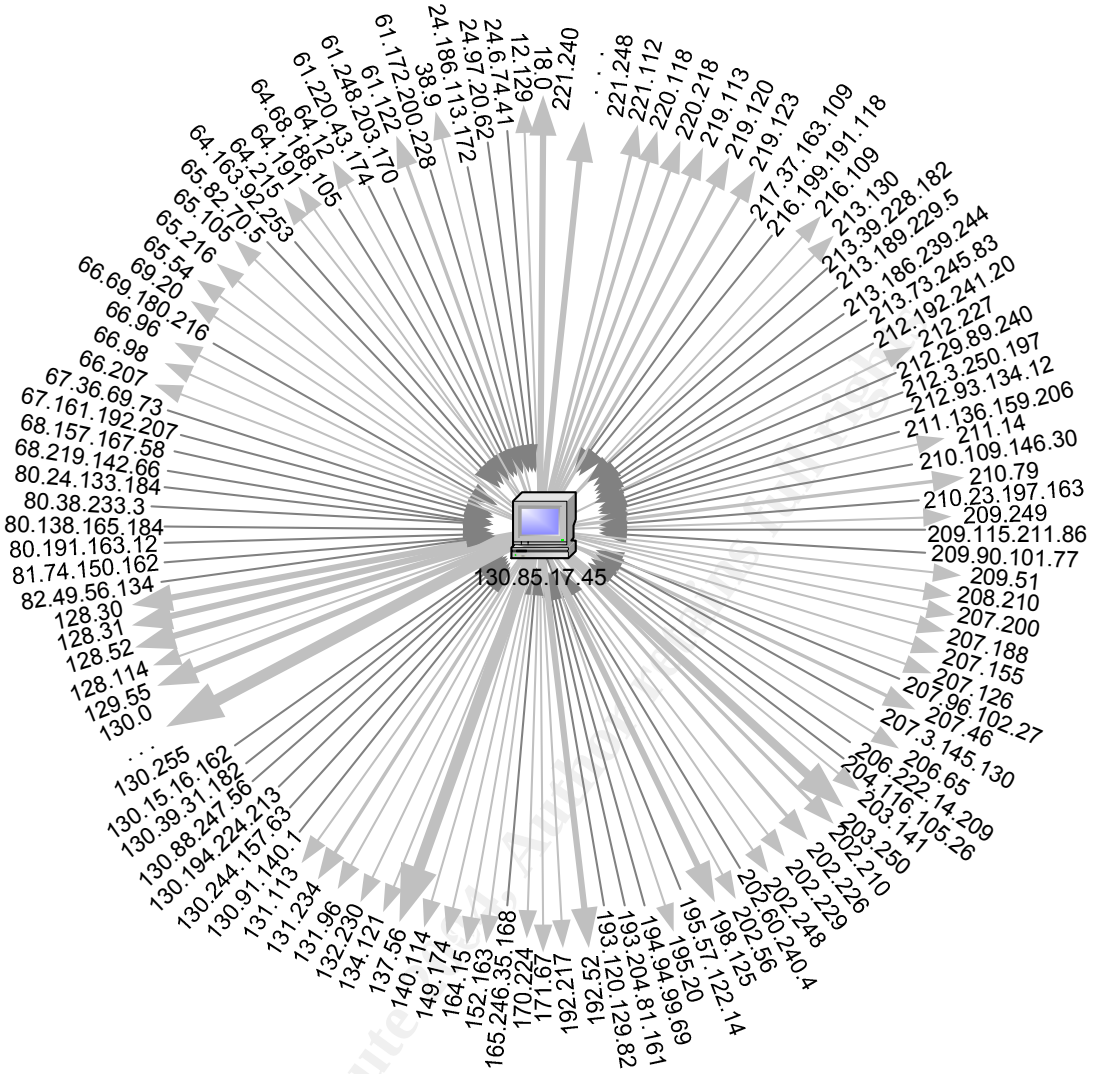
Owner		Registration Information		Contact Infromation	
descr:	WideXS Internet	**inetnum:**	212.204.214.0 - 212.204.214.127	**person**:	Peter Bosgraaf
country:	NL	**route:**	212.204.192.0/19	**address:**	WideXS
		netname:	NL-WIDEXS	**address:**	Bijlmermeerstraat 62
				address:	2131 HG Hoofddorp
				address:	The Netherlands
				phone:	+31 23 5698070
				fax-no:	+31 23 5698099
				e-mail:	peter@widexs.nl
				notify:	peter@widexs.nl
				remarks	----------
				remarks	E-mail is the preferred contact method!
				remarks	----------
				remarks	abuse@widexs.nl for abuse notifications
				remarks	hostmaster@widexs.nl for hostmaster notifications
				remarks	support@widexs.nl for customer service
				remarks	----------

This address did not appear as an attacker in the DShield database

4 Link Diagram

The host used in the link diagram `130.85.17.45` is suspected of being infected by the "W32.Mockbot.A.Worm". This diagram is based on the 1,179,273 alerts associated with this host within the scan data only. There where 51 hosts that targeted `130.85.17.45` while it targeted 142,820 individual hosts, across 52,813 C class networks or 327 B class networks.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



5 Conclusions

The overall health of the University network is satisfactory considering its size. However, some issues that have been identified do need addressing, the details are within this analysis. The University needs to prioritise its resources combating these security issues for the network to remain at its current state. The immediate concern is with the "W32.Mockbot.A.Worm" that appears to have compromised several hosts. Various P2P programs are in use on the University network, the use of these programs needs to be determined and if deemed inappropriate the identified hosts should be cleaned of this software.

6 Defensive Recommendations

The amount of data is almost overwhelming until you devise a method to tackle the task. This analysis tried to look at obviously bad behaviour, while also looking for issues that can easily be solved to reduce the false positive rate, to help make subsequent analysis more meaningful. Much of the alert data is related to false positives, the set of snort rules used appear to be quite generic, and mostly out of date, they should be tuned to remove easily identified false positives.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Some things that can be used to address the Universities security posture are:

1. Utilise the host specific rules to identify scanning hosts as described in 2.2.6;
2. Provide an Intranet server related to security issues
    a. Describe the latest network issues and general security tips;
    b. Provide the latest patches for various operating systems, or links to their location;
    c. Links to free firewalls for various operating systems;
    d. Links to anti virus software for various operating systems.
3. Scan the network environment for vulnerable hosts using nessus, I understand that this recommendation may not be appropriate in a University network but it would help identify insecure or incorrectly patched systems.

These things may help reduce the quantity of viruses, worms and Trojans spreading throughout the University network.

7 Appendix

7.1 Appendix A – The process

There is not the space to detail all that was done during the analysis process, this appendix shows at a minimum how the main tables in sections 2.1 Alert Summary and 2.3 Scans Summary were created.

7.1.1 Table 9 - Alerts by Network Segment

To create the table I started with a basic while loop:

```
let i=0 ; while [ $i -le 256 ] ; do <<something to repeat>> ; let i=i+1; done
```

To determine the count of alerts sourced from the specific net range:

```
awk -F: '{print $5}' alert.misc3 | grep "^130\.85\.$i\.[0-9]\+" | cut -d"." -f 1,2,3 | sort
   | uniq –c
```

To calculate the destination count only the "`print`" was changed to "`$7`" while the scan data had the file change to scans.clean.csv and the "`print`" statement changed accordingly. The OOS data was a little more complex as I needed to incorporate the grep command introduced in section 1.4.3, change the file name, and the "`print`" statements.

7.1.2 Table 10 - Alerts by Type

The frequency of alert data was determined by:

```
awk -F: '{ print $4 }' alert.misc3 | sort | uniq -c | sort -rn >> alert.misc3.highest-alert-
   types
```

7.1.3 Table 11 - Top 20 Alert Destination Ports

The totals where determined using this command:

```
awk -F: '{print $8}' alert.misc3 | sort | uniq -c | sort -rn > alert.misc3.ports
```

To determine the unique alerts that targeted port 80:

```
awk -F: '{print $8":"$4}' alert.misc3 | grep "^80:" | awk -F: '{ print $2 }' | sort | uniq -
   c | sort -rn > alert.misc3.csv.80.alerts
```

But to make the task much easier I create a file named "`portlist1`" that contained the top 20 port numbers and ran all the ports through a loop:

```
for i in `cat portlist1` ; do echo "starting $i" ; awk -F: '{print $8":"$4}' alert.misc3 |
   grep "^$i:" | awk -F: '{ print $2 }' | sort | uniq -c | sort -rn >
   alert.misc3.csv.$i.alerts ; echo "finished $i" ; done
```

Using the same loop structure I used this body to determine the unique external source addresses targeting said port:

```
awk -F: '{print $8":"$5}' alert.misc3 | grep "^$i:" | awk -F: '{ print $2 }' | grep -v
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   alert.misc3.csv.$i.external.source
```

And the unique internal IP addresses that received alerts to each port:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
awk -F: '{print $8":"$7}' alert.misc3 | grep "^$i:" | awk -F: '{ print $2 }' | grep
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   alert.misc3.csv.$i.internal.destination
```

Unique internal source addresses targeting said port:

```
awk -F: '{print $8":"$5}' alert.misc3 | grep "^$i:" | awk -F: '{ print $2 }' | grep
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   alert.misc3.csv.$i.internal.source
```

And the unique external target IP addresses:

```
awk -F: '{print $8":"$7}' alert.misc3 | grep "^$i:" | awk -F: '{ print $2 }' | grep -v
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   alert.misc3.csv.$i.external.destination
```

The totals where calculated for each port using the UNIX word count command "`wc`":

```
wc -l alert.misc3.csv.$i.alerts > alert.misc3.csv.$i.alerts.wc
wc -l alert.misc3.csv.$i.external.source > alert.misc3.csv.$i.external.source.wc
wc -l alert.misc3.csv.$i.internal.destination > alert.misc3.csv.$i.internal.destination.wc
wc -l alert.misc3.csv.$i.internal.source > alert.misc3.csv.$i.internal.source.wc
wc -l alert.misc3.csv.$i.external.destination > alert.misc3.csv.$i.external.destination.wc
```

The description of known uses for the given port was from information obtained through http://www.iana.org/assignments/port-numbers, and http://www.dshield.org/

7.1.4 Table 12 - Alert Reflexive Port Combinations

Firstly to determine if there where any reflexive IP address combinations I used this:

```
awk -F: '{print $5":"$6}' alert.misc3 | grep "^\([0-9]\+\.[0-9]\+\.[0-9]\+\.[0-9]\+\):\1$" |
   sort | uniq -c | sort -rn
```

This listed the alerts with the source and destination ports the same and their frequency.

```
awk -F: '{print $6":"$8}' alert.misc3 | grep "^\([0-9]\+\):\1$" | sort | uniq -c | sort -rn
```

Using our loop and a file containing the identified ports "`reflex.port.list`" these commands populated the rest of the table

```
awk -F: '{print $6":"$8":"$5}' alert.misc3 | grep "^$i:$i:" | cut -d":" -f 3 | sort | uniq -
   c | sort -rn > reflex.port.$i.source
awk -F: '{print $6":"$8":"$7}' alert.misc3 | grep "^$i:$i:" | cut -d":" -f 3 | sort | uniq -
   c | sort -rn > reflex.port.$i.destination
wc -l reflex.port.$i.source > reflex.port.$i.source.wc
wc -l reflex.port.$i.destination > reflex.port.$i.destination.wc
```

7.1.5 Table 13 - Top 10 Alert External Talkers

The external source IP address totals where determined by:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep -v "^130\.85\.[0-9]\+\.[0-9]\+"
   | awk -F: '{ print $1 }' | sort | uniq -c | sort -rn > alert.misc3.external.talkers
```

The FQDN was determined using `dig -x <ip address>`

The unique destination IP addresses was performed using the `for` loop mentioned in section 2.1.2 using a file "`iplist1`" containing all the IP addresses for the next 4 tables:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep "^$i" | awk -F: '{ print $3 }'
   | sort | uniq -c | sort -rn > alert.misc3.csv.$i
```

Then to determine the unique destination ports from each source host:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep "^$i" | awk -F: '{ print $4 }'
   | sort | uniq -c | sort -rn > alert.misc3.csv.$i.ports
```

To determine how many unique entries there where I counted the lines in each file:

```
wc -l alert.misc3.csv.$i > alert.misc3.csv.$i.wc
wc -l alert.misc3.csv.$i.ports > alert.misc3.csv.$i.ports.wc
```

7.1.6 Table 14 - Top 10 Alert Internal Targets

To determine the internal targets:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep -v '^130\.85\.[0-9]\+\.[0-9]\+'
   | awk -F: '{ print $3 }' | sort | uniq -c | sort -rn > alert.misc3.internal.targets
```

The process of populating the table was similar to that of Table 13 except for one key difference common to both Table 14 and Table 16. To determine what IP addresses and ports targeted `130.85.30.4` along with the frequency I reordered the elements in the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

source alert file from `src-ip:src-port:dst-ip:dst-port` to `dst-ip:dst-port:src-ip:dst-port` with the leading "`awk`" command, otherwise everything else was the same.

```
awk -F: '{ print $7":"$8":"$5":"$6}'
```

7.1.7 Table 15 - Top 10 Alert Internal Talkers

To determine the frequency of internal talkers:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep "^130\.85\.[0-9]\+\.[0-9]\+" |
   awk -F: '{ print $1 }' | sort | uniq -c | sort -rn > alert.misc3.internal.talkers
```

The process of populating this table was similar to that of Table 13.

7.1.8 Table 16 - Top 10 Alert External Targets

External targets list and the frequency were determined by:

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep '^130\.85\.[0-9]\+\.[0-9]\+' |
   awk -F: '{ print $3 }' | sort | uniq -c | sort -rn > alert.misc3.external.targets
```

The process of populating this table was similar to that of Table 13 except the "`awk`" change highlighted in section 7.1.6

7.1.9 Table 17 - Top 10 Alert Types from External Hosts

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep -v "^130\.85\.[0-9]\+\.[0-9]\+"
   | awk -F: '{ print $5 }' | sort | uniq -c | sort -rn > alert.misc3.external.alert.types
```

7.1.10 Table 18 - Top 10 Alert Types from Internal Hosts

```
awk -F: '{ print $5":"$6":"$7":"$8":"$4}' alert.misc3 | grep "^130\.85\.[0-9]\+\.[0-9]\+" |
   awk -F: '{ print $5 }' | sort | uniq -c | sort -rn > alert.misc3.internal.alert.types
```

7.1.11 Table 28 - Top 20 Scan Types

```
awk -F: '{ print $8 }' scans.clean.csv | sort | uniq -c | sort -rn >>
   scans.clean.csv.scans.by.type
```

7.1.12 Table 29 - Top 20 Scan Destination Ports

To determine the frequency of alert directed to each port:

```
awk -F: '{print $7}' scans.clean.csv | sort | uniq -c | sort -rn > scans.clean.csv.ports
```

After creating a file named "`portlist1`" containing all 20 port numbers and using the same loop as described in section 2.1.2 this command identified the unique alerts related to each port

```
awk -F: '{print $7":"$8}' scans.clean.csv | grep "^$i:" | awk -F: '{ print $2 }' | sort |
   uniq -c | sort -rn > scans.clean.csv.$i.scans
```

This determined unique external source addresses targeting said port

```
awk -F: '{print $7":"$4}' scans.clean.csv | grep "^$i:" | awk -F: '{ print $2 }' | grep -v
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   scans.clean.csv.$i.external.source
```

And the unique internal IP addresses that received alerts to each port:

```
awk -F: '{print $7":"$6}' scans.clean.csv | grep "^$i:" | awk -F: '{ print $2 }' | grep
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   scans.clean.csv.$i.internal.destination
```

Unique internal source addresses targeting said port:

```
awk -F: '{print $7":"$4}' scans.clean.csv | grep "^$i:" | awk -F: '{ print $2 }' | grep
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   scans.clean.csv.$i.internal.source
```

And the unique external IP addresses that received alerts directed to each port

```
awk -F: '{print $7":"$6}' scans.clean.csv | grep "^$i:" | awk -F: '{ print $2 }' | grep -v
   "^130\.85\.[0-9]\+\.[0-9]\+" | sort | uniq -c | sort -rn >
   scans.clean.csv.$i.external.destination
```

Then a "`wc`" was performed on each port number

```
wc -l scans.clean.csv.$i.scans > scans.clean.csv.$i.scans.wc
wc -l scans.clean.csv.$i.external.source > scans.clean.csv.$i.external.source.wc
wc -l scans.clean.csv.$i.internal.destination > scans.clean.csv.$i.internal.destination.wc
wc -l scans.clean.csv.$i.internal.source > scans.clean.csv.$i.internal.source.wc
wc -l scans.clean.csv.$i.external.destination > scans.clean.csv.$i.external.destination.wc
```

The description of known uses for the given port was from information obtained through http://www.iana.org/assignments/port-numbers, and http://www.dshield.org/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

7.1.13 Table 30 - Scan Reflexive Port Combinations

Firstly, to determine if there where any reflexive IP address combinations I used this:

```
awk -F: '{print $4":"$6}' scans.clean.csv | grep "^\([0-9]\+\.[0-9]\+\.[0-9]\+\.[0-
   9]\+\):\1$" | sort | uniq -c | sort -rn
```

This listed all the alerts that had the source and destination ports the same and frequency.

```
awk -F: '{print $5":"$7}' scans.clean.csv | grep "^\([0-9]\+\):\1$" | sort | uniq -c | sort
   -rn
```

Using our loop and a file containing the identified ports "`reflex.port.list`" these commands populated the rest of the table

```
awk -F: '{print $5":"$7":"$4}' scans.clean.csv | grep "^$i:$i:" | cut -d":" -f 3 | sort |
   uniq -c | sort -rn > reflex.port.$i.source
awk -F: '{print $5":"$7":"$6}' scans.clean.csv | grep "^$i:$i:" | cut -d":" -f 3 | sort |
   uniq -c | sort -rn > reflex.port.$i.destination
wc -l reflex.port.$i.source > reflex.port.$i.source.wc
wc -l reflex.port.$i.destination > reflex.port.$i.destination.wc
```

7.1.14 Table 31 - Top 10 Scan External Talkers

To determine the frequency of external source IP addresses:

```
awk -F: '{ print $4 }' scans.clean.csv | sort | uniq -c | sort -rn | grep -v '130\.85\.[0-
   9]\+\.[0-9]\+' > scans.clean.csv.external.talkers
```

FQDN determined using `dig -x <ip address>`

Using our `for` loop and using a file "`iplist1`" containing all the IP addresses for the next four tables this line helped determine the destination IP address and the count to each:

```
awk -F: '{ print $4":"$5":"$6":"$7":"$8}' scans.clean.csv | grep "^$i" | awk -F: '{ print $3
   }' | sort | uniq -c | sort -rn > scans.clean.csv.$i
```

To determine the unique destination ports from each source host:

```
awk -F: '{ print $4":"$5":"$6":"$7":"$8}' scans.clean.csv | grep "^$i" | awk -F: '{ print $4
   }' | sort | uniq -c | sort -rn > scans.clean.csv.$i.ports
```

Ports with counts greater than a set number are in bold to give a sense of what ports have been specifically targeted, ports greater than 1023 are considered ephemeral ports. Lastly to determine how many unique entries there where I counted each line in the files:

```
wc -l scans.clean.csv.$i > scans.clean.csv.$i.wc
wc -l scans.clean.csv.$i.ports > scans.clean.csv.$i.ports.wc
```

7.1.15 Table 32 - Top 10 Scan Internal Targets

The external IP address frequency was determined:

```
awk -F: '{ print $6 }' scans.clean.csv | sort | uniq -c | sort -rn | grep '130\.85\.[0-
   9]\+\.[0-9]\+' > scans.clean.csv.internal.targets
```

The process of populating this table was similar to that of Table 31 except for one key difference common to Table 32 and Table 34. To determine what IP addresses and ports targeted `130.85.27.232` along with the frequency I reordered the elements in the source scan file from `src-ip:src-port:dst-ip:dst-port` to `dst-ip:dst-port:src-ip:dst-port` with the leading "`awk`" command otherwise the process was the same.

```
awk -F: '{ print $6":"$7":"$4":"$5}'
```

7.1.16 Table 33 - Top 10 Scan Internal Talkers

The frequency of internal IP address as sources of alerts was determined by:

```
awk -F: '{ print $4 }' scans.clean.csv | sort | uniq -c | sort -rn | grep '130\.85\.[0-
   9]\+\.[0-9]\+' > scans.clean.csv.internal.talkers
```

The process of populating this table was similar to that of Table 31.

7.1.17 Table 34 - Top 10 Scan External Targets

The frequency of external IP address as targets was determined by:

```
awk -F: '{ print $6 }' scans.clean.csv | sort | uniq -c | sort -rn | grep -v '130\.85\.[0-
   9]\+\.[0-9]\+' > scans.clean.csv.external.targets
```

The process of populating this table was similar to that of Table 31 except the "`awk`" change highlighted in section 7.1.15

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

7.1.18 Table 43 - Top 10 OOS External Talkers

To determine the frequency of external source IP addresses:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $4 }' | sort | uniq -c | sort -rn
   | grep -v '130\.85\.[0-9]\+\.[0-9]\+'
```

FQDN determined using `dig -x <ip address>`

Using our `for` loop and a file "`iplist1`" containing all the IP addresses for the next four tables this line helped determine the destination IP address and the count to each:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $4":"$5":"$6":"$7}' | grep "^$i"
   | awk -F: '{ print $3 }' | sort | uniq -c | sort -rn
```

To determine the unique destination ports from each source host:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $4":"$5":"$6":"$7}' | grep "^$i"
   | awk -F: '{ print $4 }' | sort | uniq -c | sort -rn
```

7.1.19 Table 44 - Top 10 OOS Internal Targets

The external IP address frequency was determined:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $6 }' | sort | uniq -c | sort -rn
   | grep '130\.85\.[0-9]\+\.[0-9]\+'
```

The process of populating this table was similar to that of Table 43 except for one key difference common to Table 44 and Table 46. To determine what IP addresses and ports targeted `130.85.6.7` along with the frequency I reordered the elements in the source scan file from `src-ip:src-port:dst-ip:dst-port` to `dst-ip:dst-port:src-ip:dst-port` with the leading "`awk`" command otherwise the process is the same.

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $6":"$7":"$4":"$5}' | grep "^$i"
   | awk -F: '{ print $3 }' | sort | uniq -c | sort -rn
```

To determine the unique destination ports from each source host:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $6":"$7":"$4":"$5}' | grep "^$i"
   | awk -F: '{ print $2 }' | sort | uniq -c | sort -rn
```

7.1.20 Table 45 - Top 10 OOS Internal Talkers

The frequency of internal IP address as sources of alerts was determined by:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $4 }' | sort | uniq -c | sort -rn
   | grep '130\.85\.[0-9]\+\.[0-9]\+'
```

The process of populating this table was similar to that of Table 43.

7.1.21 Table 46 - Top 10 OOS External Targets

The frequency of external IP address as targets was determined by:

```
grep '^04\/2[0-6]-[0-9]\+:[0-9]\+:[0-9]\+.[0-9]\+ ' oos_report.all.clean | sed 's/ \-> /:/g'
   | sed 's/\(:[0-9]\+.[0-9]*\) /\1:/g' | awk -F: '{ print $6 }' | sort | uniq -c | sort -rn
   | grep -v '130\.85\.[0-9]\+\.[0-9]\+'
```

The process of populating this table was similar to that of Table 44

7.2 Appendix B – References

Almost all the references are inline with the text; here are some other references used during the compilation of this paper:

Mastering Regular Expressions (2nd Edition)
Jeffrey E. F. Friedl (Published by O'Reilly)

Building Internet Firewalls (2nd Edition)
Elizabeth D. Zwicky, Simon Cooper & D. Brent Chapman

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.