An Introduction to Network Security Data Visualization

Greg Conti

1. Introduction

The Internet is teeming with network traffic, usually benign, but occasionally malicious. A recent study by the University of California at Berkeley estimated that 532,897 terabytes of information flowed across the Internet in 2002. Of that figure, approximately 92,000 terabytes were World Wide Web traffic, 440,000 terabytes were original email traffic and 274 terabytes were instant messaging traffic.[1] Given an estimate of 1 kilobyte per packet, approximately 533 trillion packets traversed the Internet. While the exact percentage of malicious packets is not known, assuming a conservative estimate of 0.1%, one can estimate the traffic in the hundreds of billions. This figure is clouded by "crud" from legitimate traffic exhibiting abnormal behavior.[2] Traditional intrusion detection systems use algorithmically implemented signature and anomoly based techniques to detect attacks. These techniques, while effective, result in an unacceptable amount of false positives and false negatives when confronted with the sheer volume of network activity.

In this white paper, I present an approach to augment the algorithmically implemented, traditional signature and anomoly based intrusion detection systems with visualization techniques. This provides multiple benefits:

- By their very nature, most zero-day (never seen before) attacks do not match existing intrusion signatures. Visualization techniques can provide clues to impending attacks and facilitate quick response analysis.
- Some stealthy attacks are invisible to traditional intrusion detection systems, but are readily visible using appropriate visualizations.
- Specific attack tools and exploits have visual fingerprints that can be used to profile attackers.
- Visualization techniques can be used that require little state and are remarkably resistant to resource consumption attacks that can incapacitate traditional intrusion detection systems.
- Both real time and historical attacks can be visualized effectively, aiding forensic analysis.
- An attacker's methodology (foot printing, scanning, enumeration, gaining access, escalating privilege, pilfering, covering tracks, creating back doors and denial of service)[3] can be seen in ways that supplement the capabilities of algorithmic intrusion detection systems.
- Some system misconfigurations are readily apparent.
- Proper visualization can make monitoring network activity much more efficient than monitoring system logs and alerts from intrusion detection systems.

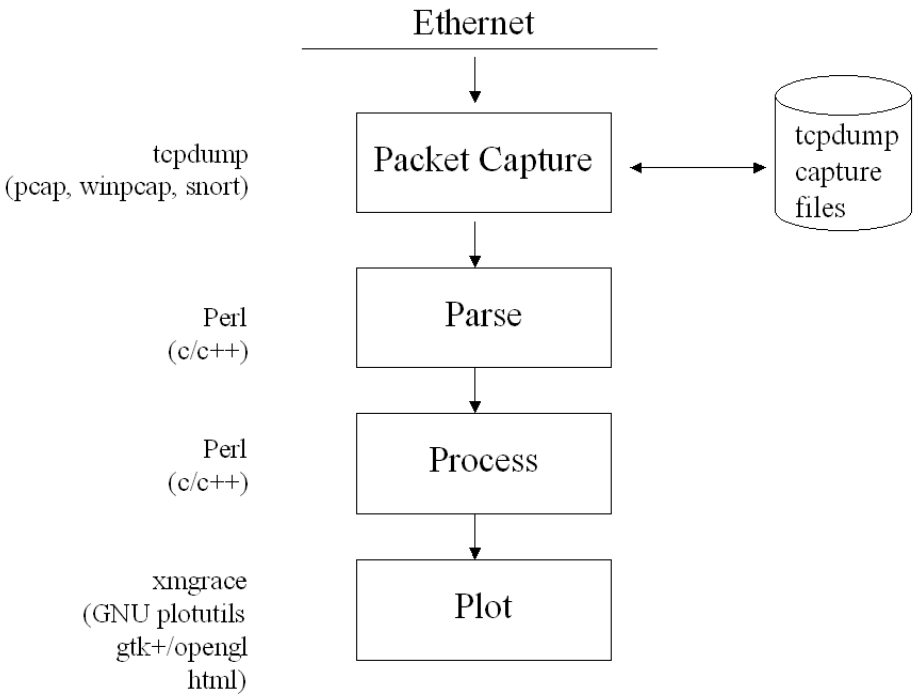
- With properly designed tools, less experienced personnel can be quickly trained to watch for attacks.
- Detecting insider threats can be easier.
- A secondary benefit is that educational Capture the Flag or Root Wars games can be made more of a spectator sport, facilitating learning.
- Properly designed tools can help reduce the search space, find suspicious patterns, enable "what if" interaction and prevent overwhelming the user.

2. Approach

Exploration of network security data is both an art and a science. I propose the following methodology when attempting to use visualization techniques to support network security:

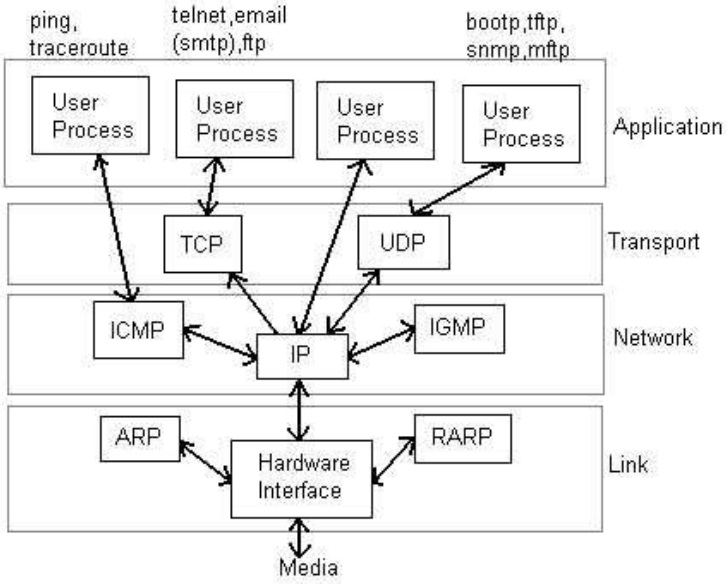
1. Examine the nature of the datasets you have available.
2. Consider the meta-data you can calculate.
3. Develop candidate combinations that you think will be interesting.
4. Clean-up and parse the data.
5. Process the data.
5. Plot and explore the data using general-purpose tools.
6. Look for insights.

The following figure shows the data flow. As an example, lets take an in depth look at the visualization of network sniffing data.



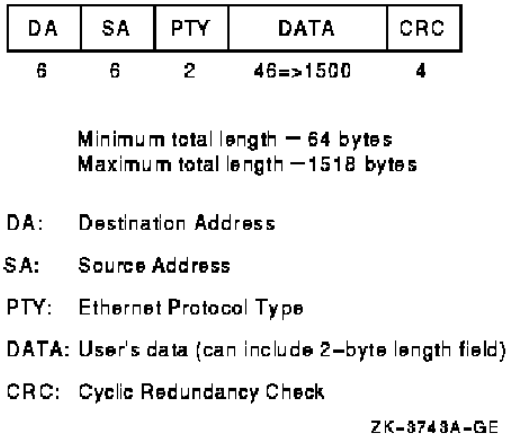
Examine the nature of the datasets you have available

In this case we are examining the header information available "on the wire." Typically this includes the link layer, network layer, transport layer and application layer. The following figure illustrates the layers.[4]

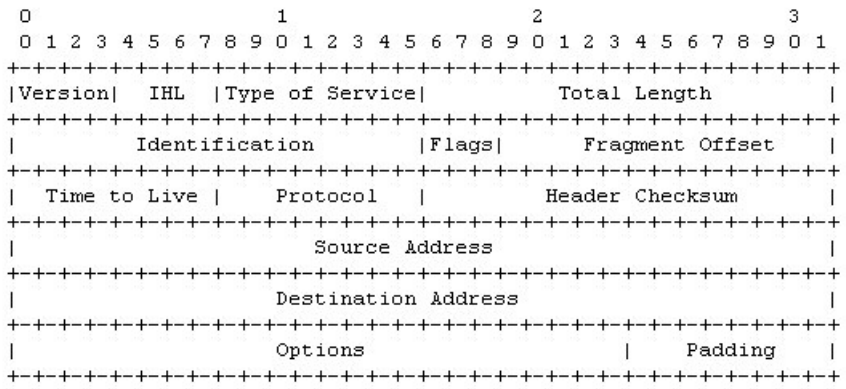


For clarity, I will focus on the link (Ethernet), network (IP) and transport layer (TCP&UDP) header information available from *tcpdump*. The following figures illustrate the available information.

Ethernet Header[5]

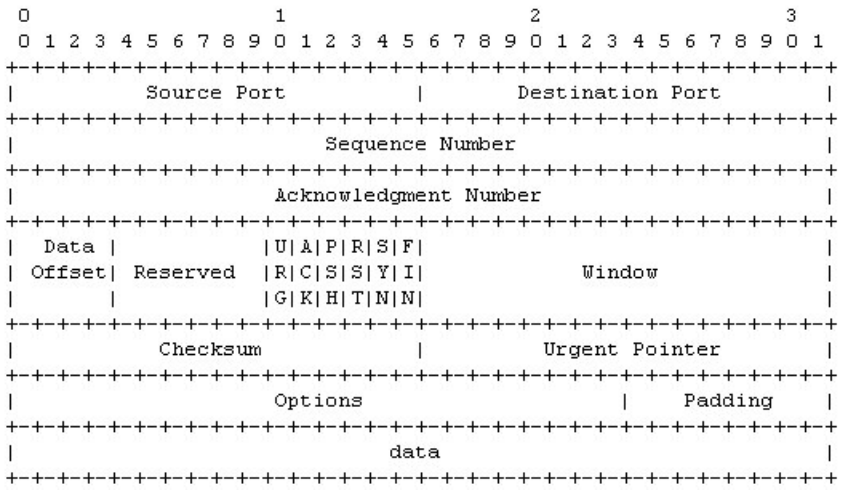


IP Header[6]



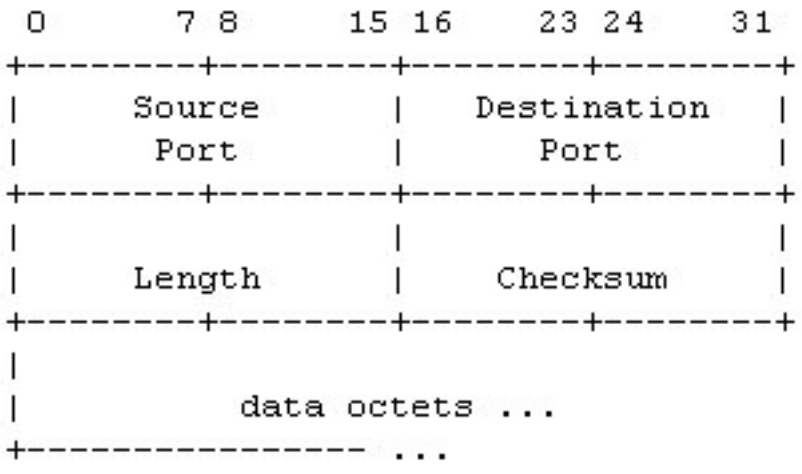
Example Internet Datagram Header

TCP Header[7]



TCP Header Format

UDP Header[8]



User Datagram Header Format

Consider the meta-data you can calculate.

Beyond the information that can be directly gained from the header information there is a great deal of meta-information that can be calculated. For example:

- Time of packet arrival
- Time between packets
- Size of packets
- Frequency of packets
- Actual checksum vs. stated checksum
- Compliance of packet with protocol
- Number of fragmented packets
- Possible fingerprint of host operating system[9]

Develop candidate combinations that you think will be interesting.

After you have become familiar with the data and meta-data you have available put together some combinations that you think would be interesting to visualize. For example:

source port vs. destination port
source IP vs. time (see full example online[10])
source IP vs. destination IP vs. destination port (see full example online)[11]
.
.
.
The options are only limited by your creativity and the characteristics of the attacks you are looking for.

For this example I will select source port and destination port.

Clean-up and parse the data.

Most general purpose visualization tools require that the data set adhere to certain formatting guidelines (e.g. tab delimited, one element per line). In this example, I will use Perl to parse and filter the output of *tcpdump* to provide me with the fields I am interested in. The example parsing script in Appendix A demonstrates this.

The output from the script looks like this:

```
09 02 01 858240 0:6:5b:4:20:14 0:5:9a:50:70:9 10.100.1.120.4532 10.100.1.120 4532 10.1.3.0.1080 10.1.3.0 1080 tcp
09 02 02 178743 0:6:5b:4:20:14 0:5:9a:50:70:9 10.100.1.120.4537 10.100.1.120 4537 10.1.3.0.1080 10.1.3.0 1080 tcp
09 03 55 638701 0:6:5b:4:20:14 0:5:9a:50:70:9 10.100.1.120.2370 10.100.1.120 2370 10.1.3.0.1080 10.1.3.0 1080 tcp
09 03 56 000873 0:6:5b:4:20:14 0:5:9a:50:70:9 10.100.1.120.2375 10.100.1.120 2375 10.1.3.0.1080 10.1.3.0 1080 tcp
```

Process the data.

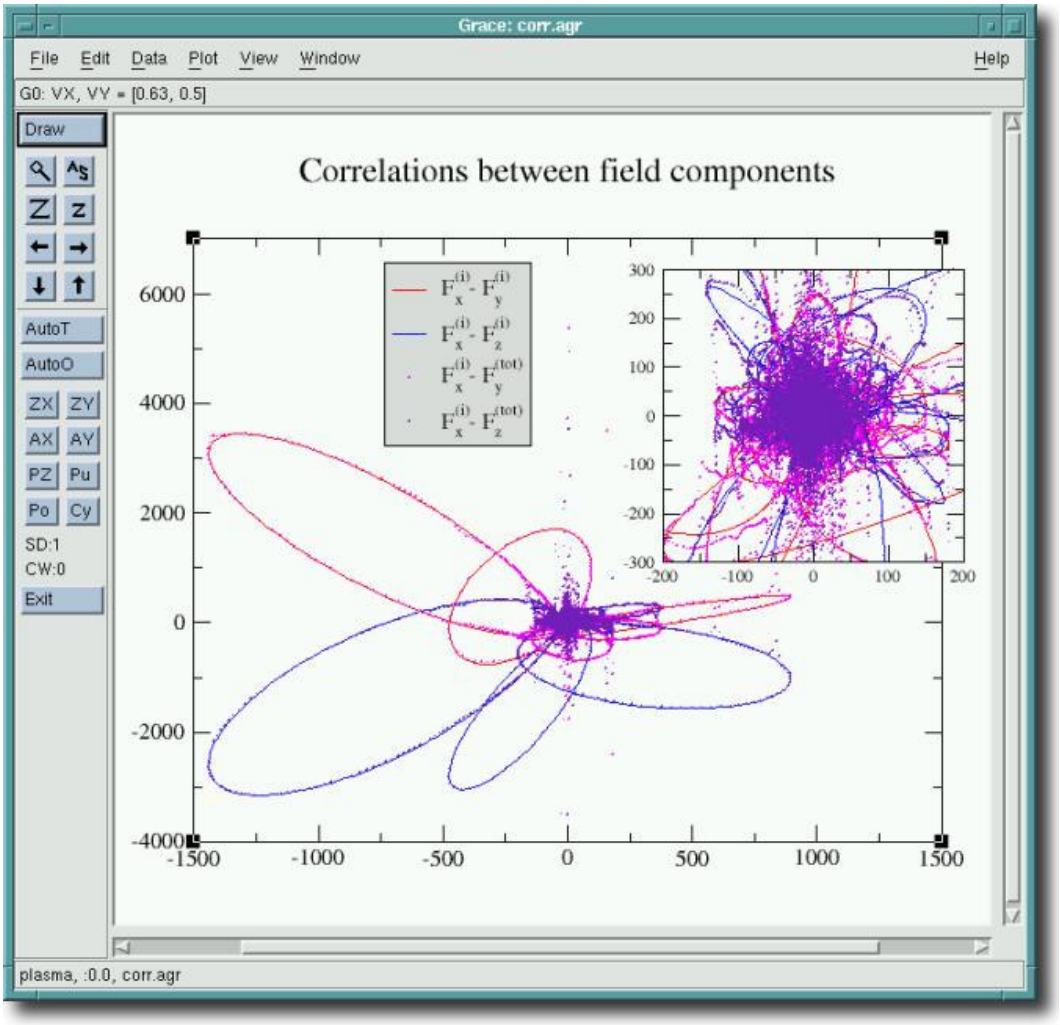
In this step you convert the data to the coordinates that you would like plotted by your graphics program. For example, you may need to convert IP addresses to 32 bit integers. In our example I want to plot the source port of all inbound packets along the left X axis (0, port number) and all packets from my monitored machine along the right X axis (1, port number). The example processing script in Appendix B demonstrates this.

The output from the script looks like this:

```
0 4532
1 1080
0 4537
1 1080
0 2370
1 1080
0 2375
1 1080
0 3532
1 8080
0 3537
1 8080
0 3958

<snipped>
```

Plot and explore the data using general-purpose tools.

At this stage you want to load the data into a general-purpose plotting tool such as gnu plot. I chose xmgrace[12] because of its self scaling and interactive capabilities (see sample screen shot below).



When using the following grace configuration file, it will load the coordinates and plot each pair as end points of a line.

```
#Demo Run
title "Port Scan Against Single Host"
subtitle "Superscan 3.0 w/Default Port Settings"
yaxis label "Port"
yaxis label place both
yaxis ticklabel place both

xaxis ticklabel off
xaxis tick major off
xaxis tick minor off

s0 color 4
s0 line type 4

autoscale
```

The following is a short howto for Grace:

To install grace...

Download RPMs (or source):
`ftp://plasma-gate.weizmann.ac.il/pub/grace/contrib/RPMS`

The files you want:
`grace-5.1.14-1.i386.rpm`
`pdflib-4.0.3-1.i386.rpm`

Install:

```
#rpm -i pdflib-4.0.3-1.i386.rpm
#rpm -i grace-5.1.14-1.i386.rpm
```

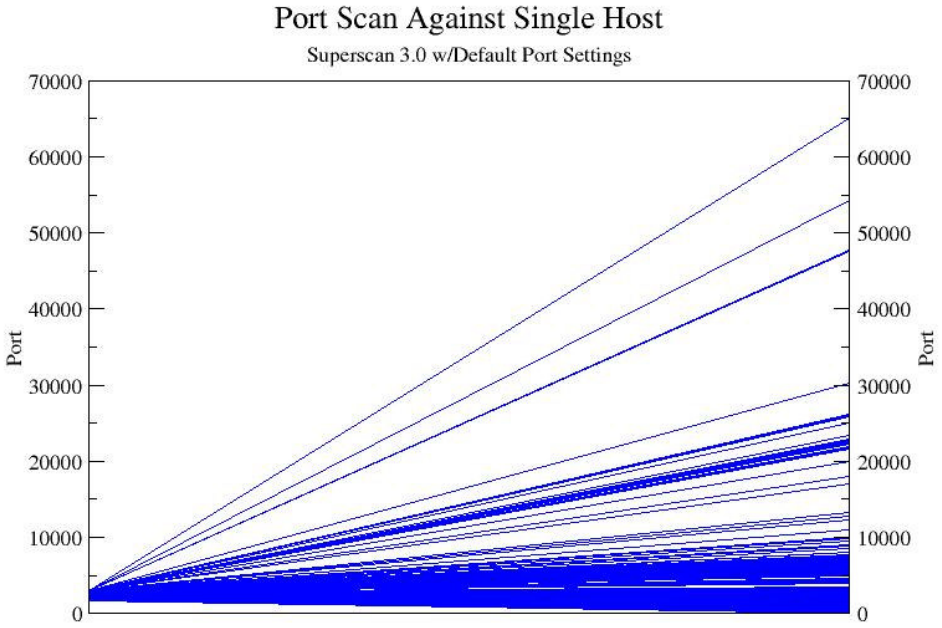
To run grace:

```
#xmgrace outfile.dat &

or

#xmgrace outfile.dat -batch grace_config.txt &
```

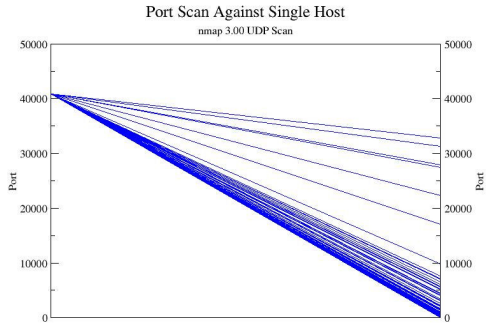
Where outfile.dat is a simple text file with a pair of X Y
coordinates per line and grace_config is the (optional) configuration file.

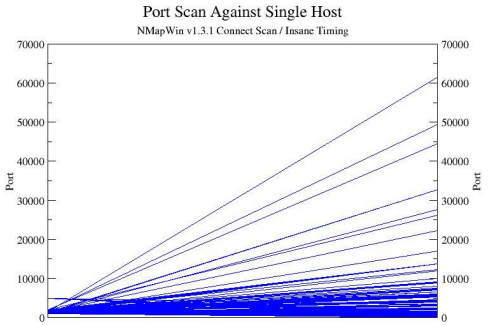
Look for insights.

During this stage use Grace to explore your data. In this example the overview of the data looked like this:



The dataset I used for this example was a port scan using Superscan 3.0.[13] By using Grace's zoom capability you can see that Superscan has a unique signature based on the range and number of ports it uses as well as the ports it targets. This signature became particularly clear when compared with scans from other tools such as an *nmap*[14] UDP scan (left) and an *NmapWin* connect scan (right).





Appendix A - Example Parser Script

```
#!/usr/bin/perl

# tcpdump parser
# parses out timestamp, src & dst mac, src & dst ip, src & dst port

# use tcpdump -l -nnqe tcp or udp | perl parse.pl
#
#  -nn don't convert host, protocols and port numbers
#      addresses to names
#  -l  use line buffered output
#  -q  quiet output
#  tcp or udp  passes only tcp or udp packets


while ($i=<STDIN>){

  #$i = <STDIN>;


  #parse input
    @fields = split / /, $i;

  #parse time field
    $time = $fields[0];
    @time_parse = split /:/, $time;

    $hours    = $time_parse[0];
    $minutes  = $time_parse[1];
    $seconds  = $time_parse[2];
    @seconds_parse = split /\./, $seconds;
    $seconds  = $seconds_parse[0];
    $useconds = $seconds_parse[1];

  #parse macs
    $src_mac = $fields[1];
    $dst_mac = $fields[2];

  #skip fields [3]

  #parse source socket
    $src_socket = $fields[4];
    $src_ip = $src_socket;
    $src_port = $src_socket;
    $src_port =~ s/^\d+\.\d+\.\d+\.\d+\.//;
    $src_len = length($src_port);
    $socket_len = length($src_ip);
    $src_ip = substr($src_ip, 0, $socket_len-$src_len-1);
                           #-1 to remove the final period


  #skip direction symbol fields[5]

  #parse destination socket
```

```
    $dst_socket = $fields[6];
    chop($dst_socket); #strip off the colon
    $dst_ip = $dst_socket;
    $dst_port = $dst_socket;
    $dst_port =~ s/^\d+\.\d+\.\d+\.\d+\.//;
    $port_len = length($dst_port);
    $socket_len = length($dst_ip);
    $dst_ip = substr($dst_ip, 0, $socket_len-$port_len-1);
                          #-1 to remove the final period

  #parse packet type (tcp||udp)
    $ip_packet_type = $fields[7];

  #output
    print "$hours $minutes $seconds $useconds ";
    print "$src_mac $dst_mac ";
    print "$src_socket $src_ip $src_port ";
    print "$dst_socket $dst_ip $dst_port ";
    print "$ip_packet_type";
    print "\n";

} # main while loop
```

Appendix B - Example Analysis Script

```
#!/usr/bin/perl

# takes output from parse.pl script

while ($i=<STDIN>){

  #parse input
    @fields = split / /, $i;

  $hours          = $fields[0];
  $minutes        = $fields[1];
  $seconds        = $fields[2];
  $useconds       = $fields[3];
  $src_mac        = $fields[4];
  $dst_mac        = $fields[5];
  $src_socket     = $fields[6];
  $src_ip         = $fields[7];
  $src_port       = $fields[8];
  $dst_socket     = $fields[9];
  $dst_ip         = $fields[10];
  $dst_port       = $fields[11];
  $ip_packet_type = $fields[12];

  #print selected output
    #print "$hours $minutes $seconds $useconds ";
    #print "$src_mac $dst_mac ";
    #print "$src_socket $src_ip $src_port ";
    #print "$dst_socket $dst_ip $dst_port ";
    #print "$ip_packet_type";
    #print "\n";

  #You will need to customize the $src_ip lines
  #to let the program know what IP address (or
  #range) you want displayed on the left or
  #right side of the graph

  #to select one IP as the target use
  #if ($src_ip eq "192.168.1.250")...
  #else...

  if ($src_ip =~ m/^10\.100\./) { #I am the target
       print "0 $src_port \n"; #0 x coord and $src_port y coord
       print "1 $dst_port \n"; #1 x coord and $src_port y coord
    } elsif ($src_ip =~ m/^10\.1\./) {#I am the destination
      print "0 $dst_port \n";
      print "1 $src_port \n";
    }

} # main while loop
```

Bibliography

1 http://www.sims.berkeley.edu/research/projects/how-much-info-2003/

2 Paxson, Vern; “Bro: A System for Detecting Network Intruders in Real-Time;” http://www.icir.org/vern/papers/bro-CN99.html, last accessed 20 February 2004.

3 Hacking Exposed

4 http://ai3.asti.dost.gov.ph/sat/levels.jpg

5 http://www.itec.suny.edu/scsys/vms/OVMSDOC073/V73/6136/ZK-3743A.gif

6 http://www.ietf.org/rfc/rfc0791.txt

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

8 http://www.ietf.org/rfc/rfc0768.txt

9 http://www.honeynet.org/papers/finger/

10 http://www.mts.jhu.edu/~marchette/ID04/homework1.txt

11 http://www.cs.hut.fi/~jtjuslin/OREILLY_v2.pdf

12 http://plasma-gate.weizmann.ac.il/Grace/

13 http://www.foundstone.com/

14 http://www.insecure.org/nmap/