2 Intrusion Detection Systems

🗀 http://www.asecuritysite.com/security/information/chapter02

2.1 Objectives

The key objectives of this unit are to:

- Provide an overview of the requirement for Intrusion Detection Systems (IDSs), and where they are used.
- Define a practical implementation of IDS using Snort.
- Outline some typical detection procedures, such as for ping sweeps.

2.2 Introduction

In Chapter 1 the concept of defence-in-depth was discussed, where a defence system has many layers of defence (Figure 2.1). Unfortunately, as in military systems, it is not always possible to protect using front-line defences, even if there are multiple layers of them, against breaches in security (Figure 2.2). This can be because an intruder has found a weakness within the security barriers, or because the intruder has actually managed to physically locate themselves within the trusted areas. Thus all the gateway firewalls and DMZ's cannot protect against an intruder once they have managed to base themselves physically or locally within a network. Along with this, most security systems can only guard against known types of attacks, such as in detecting known viruses. A particular problem is when new types of attacks occur, as these are more difficult to defend against. Thus a key factor is identifying *threats*, and how to mitigate against them. Many organisations are now rehearsing plans on how to cope with these threats, and have contingency plans. Unfortunately many other organisations have no plans for given threats, and these are the ones that are in most danger of a damaging attack.

As in military systems, an allied force would setup spies whose task it is to detect intrusions, and any covert activities. Figure 2.3 illustrates this concept, where intrusion detection agents are used to listen to network traffic, and network/user activity to try and detect any breaches in security.

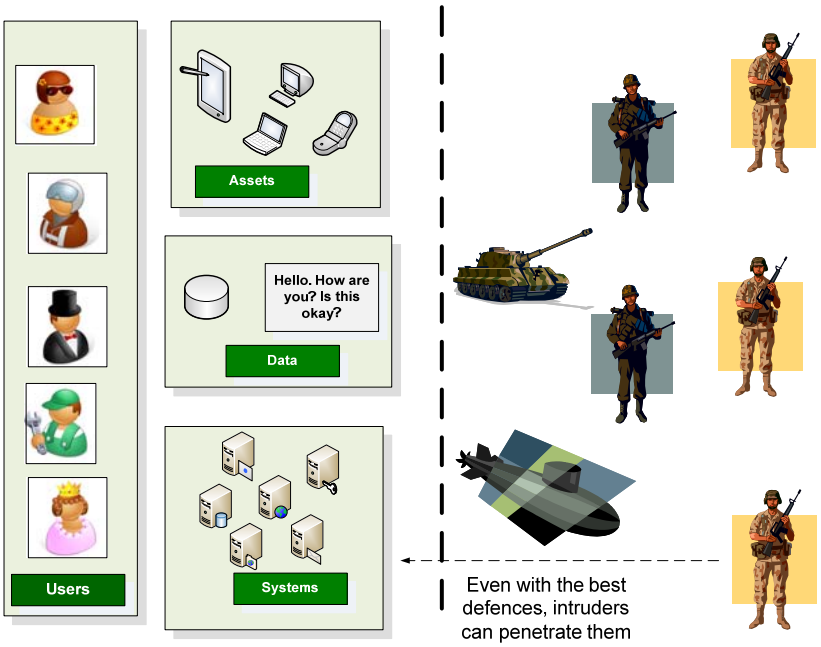


Figure 2.1 Network security

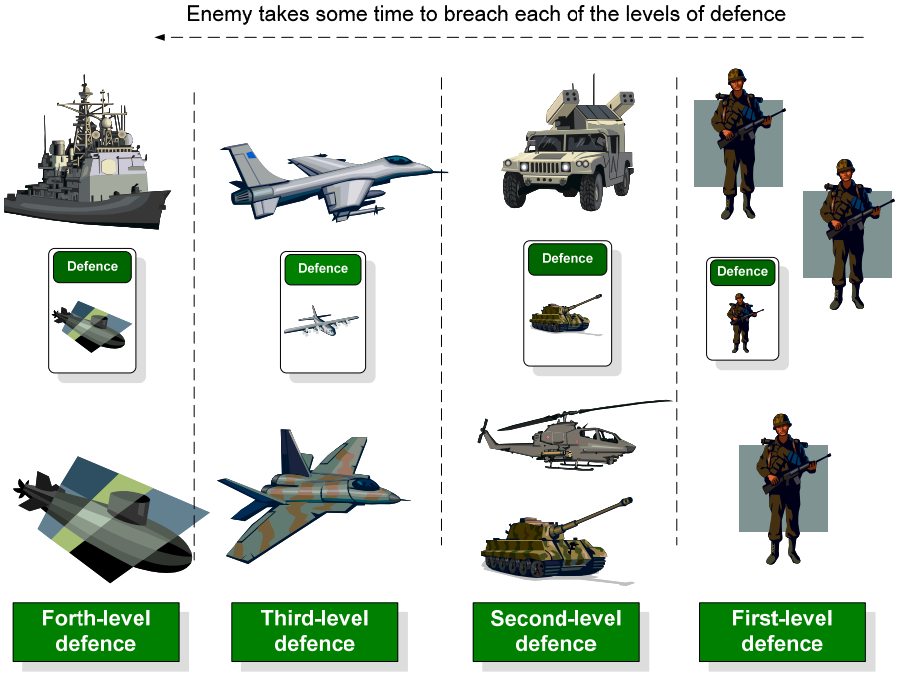


Figure 2.2 Network security

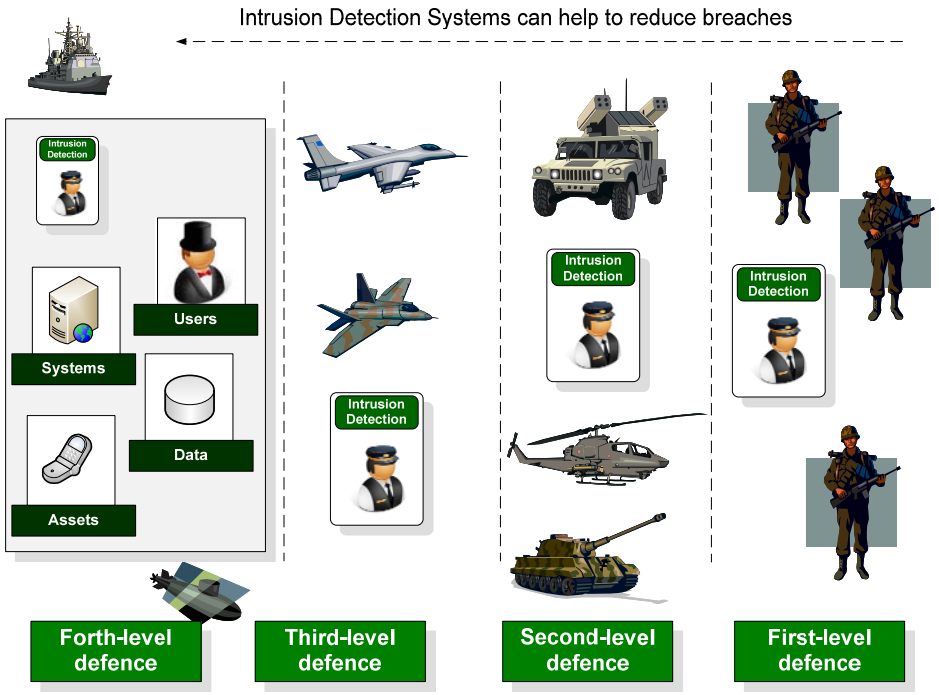


Figure 2.3 Intrusion detection

Most users think that the major threat for organisational security is that of the external intruder, such as the 'script kiddie' who typically works from home with a remote connection and who wants to do damage to the system for the glory of it. Unfortunately, this is only one part of security, as there are many other threats, from both from inside and outside the network. Thus gateway bastions, such as perimeter routers, can never been seen as an effective method of stopping network intrusions. Figure 2.4 outlines some of the threats which exist, from both inside and outside the network. These include: data stealing; personal abuse; worms/viruses; DDoS (Distributed Denial-of-Service); fraud; and terrorism. It is thus important that intrusion detection and logging agents are placed around the network, and on hosts, in order that an intrusion can be detected, or, at least, the information on the intrusion is gained for future defences (Figure 2.5).

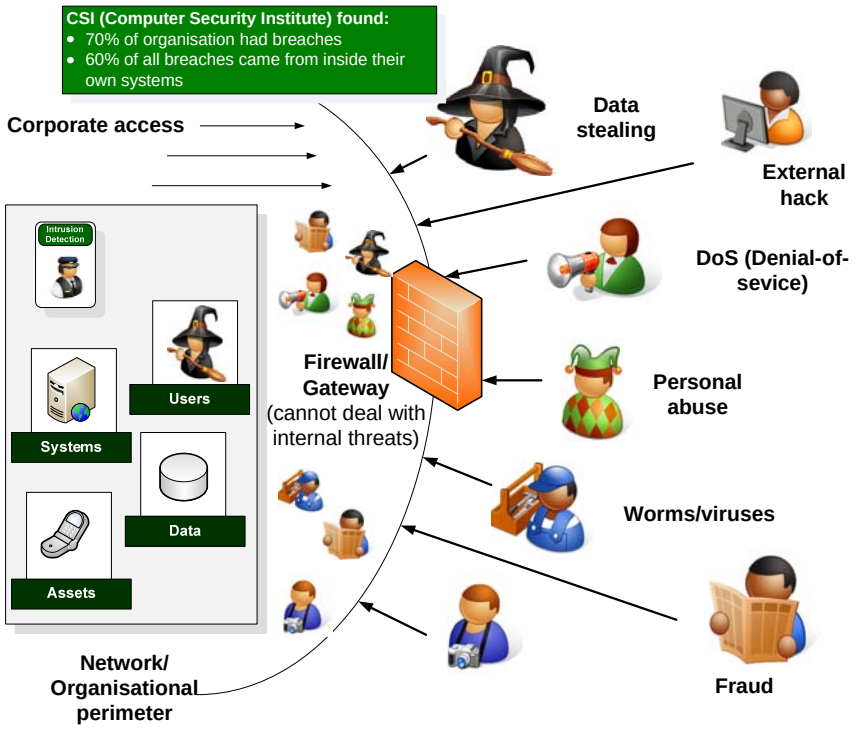


Figure 2.4 Network threats

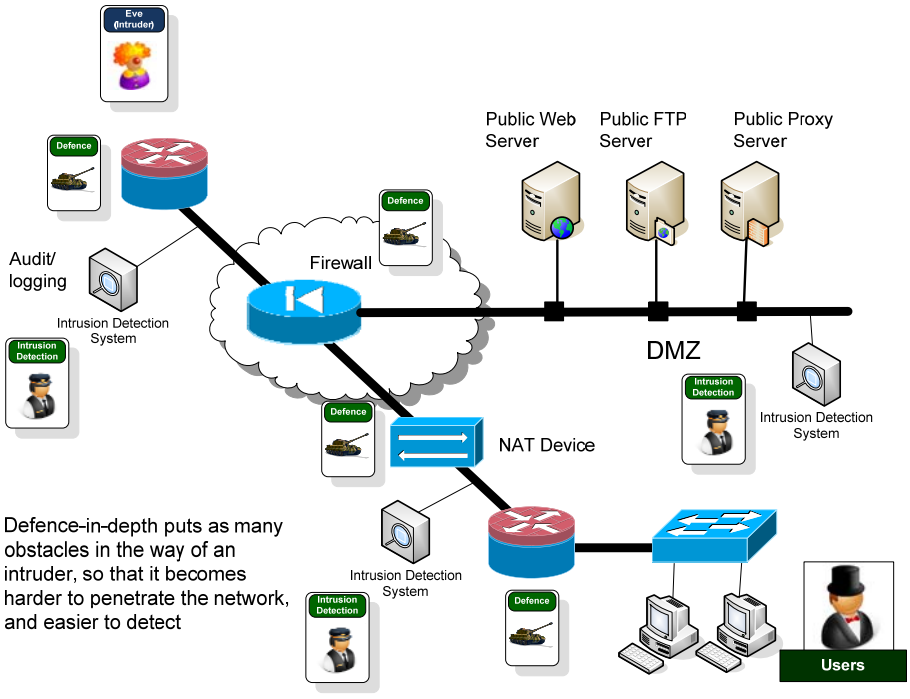


Figure 2.5 Intrusion detection agents

2.3 Types of intrusion

There are two main types of intrusion detection:

- **Misuse (Signature-based) Detection**. This type of IDS attempts to model threats with specific well-defined patterns, and then scans for occurrences of these. Typi-

cal types of misuse detection includes: the propagation of well-known viruses; and worm propagation. Its main disadvantage is that it struggles to detect new attacks, as these are likely to have signatures which do not match current attacks. This method is also good at detecting script-based attacks, such as using NMAP to scan the hosts on a network, as the scripts tend to have a fairly well defined operation.

- **Anomaly Detection**. This type of IDS assumes that abnormal behaviour by a user/device can be correlated with an intrusion. Its advantage is that it can typically react to new attacks, but can often struggle to detect variants of known threats, particularly if they fit into the normal usage pattern of a user. Another problem is that they can be overcome if they the intruder mimics the normal behavioural pattern of users/devices. This type of detection is good for human-type threats, such as with fraud, where an anomaly detector can pick-up changes in user behaviour, which is often a sign of potential fraud. Typically anomoly classifications relate to user anomolies (such as a change in user behaviour), host anomolies (such as a change in machine operation, such as increased CPU usage, and an increased number of system processes) and network anomolies (such as a change in network traffic, such as an increase in FTP traffic).

The main types of intrusion detection systems are:

- **Network intrusion detection systems** (NIDS). These monitor data packets on the network and try to determine an intrusion based on network traffic. They can either be host-based, where it runs on a host, or network-based, where they can listen to network traffic using a hub, router or probe. **Snort** is a good example of a NIDS, and is freely available for most operating systems.
- **System integrity verifiers** (SIV). These monitor system files to determine if an intruder has changed them, such as with a backdoor attack. A good example of this is **Tripwire**. They can also watch other key system components, such as the Windows registry and for root/administrator level privileges.
- **Log file monitors** (LFM). These monitor log files which are generated by application servers and networked services, and look for key patterns of change. **Swatch** is a good example of an LFM.
- **User profiling**. This involves monitoring user behaviour, where the system checks for normal user behaviour against the current user behaviour. Any anomalies, or differences from the norm, could point to an intrusion.
- **Honey pots**. This is where an administrator places a host on the network which is prone to attack, such as: having weak or dummy passwords; an unpatched operating system; or have TCP server ports open for connection. The honey pot is thus used to attract an intruder, and detect the intrusion at any early stage. Some advanced honey pots try and mimic the required responses of an attacked host, but not actually implement the attack.

2.4 Attack patterns

It is important to know the main stages of an intrusion, so that they can be detected at an early phase, and to overcome them before they can do any damage. Basically an

intrusion typically goes through alert phases from yellow, which shows some signs of a potential threat, to red, which involves the potential stealing of data or some form of abuse. The main phases are defined in Figure 2.6.

Often it takes some time for an intruder to profit from their activities, and it is important to put in as many obstacles as possible to slow down their activity. The slower the intrusion, the more chance there is in detecting the activates, and thus in thwarting them. Figure 2.6 shows a typical sequence of intrusion, which goes from a yellow alert (on the outside reconnaissance) to a red alert (for the profit phase).

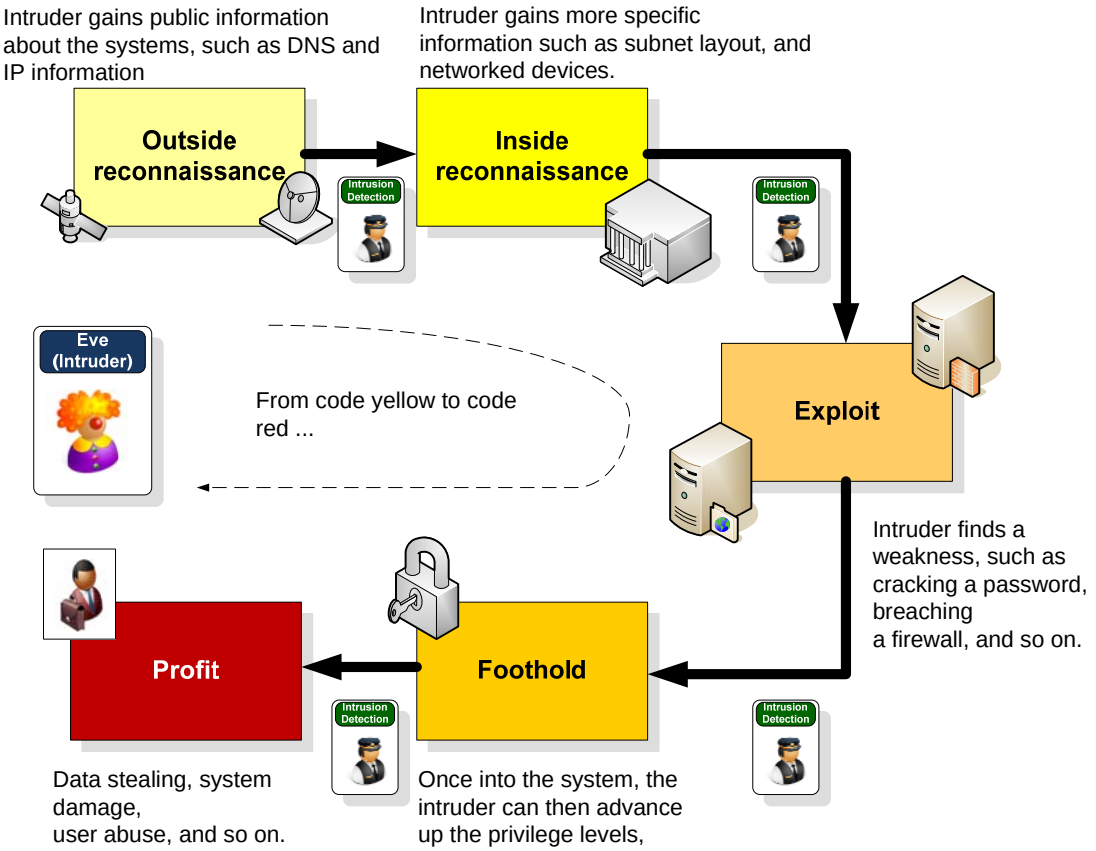


Figure 2.6 Intrusion pattern

Initially an intruder might gain information from outside the network, such as determining network addresses, or domain names. There are, unfortunately, many databases which contain this type of information, as the Internet is a global network, and organisations must register their systems for network addresses and domain names. Once gained, the intruder could move into an internal reconnaissance phase, where more specific information could be gained, such as determining the location of firewalls, subnetworks, network layouts, host/server locations, and so on. It is thus important that this type of activity is detected, as it is typically a sign of some form of future intrusion. Key features could be things such as:

- A scan of network addresses for a range of hosts on a given subnetwork (ping sweep).
- A scan of open TCP ports for a range of hosts on a given subnetwork (port scan).

- A scan of a specific TCP port for a range of hosts on a given subnetwork (port sweep).
- An interrogation of the configuration of network devices.
- Accessing systems configuration files, such as ones which contain user names and passwords.

Once the intruder has managed to gain information from the internal network, they may then use this information to gain a foothold, from which they can exploit. Example of this may be:

- Hijacking a user ID which has a default password (such as for the password of **default** or **password**), and then using this to move up the levels of privilege on a system. Often the administrator has the highest privileges on the system, but is normally secured with a strong password. An intruder, though, who gains a foothold on the system, normally through a lower-level account, could then gleam more information, and move up through the privilege hierarchy.
- Using software flaws to exploit weaknesses, and gain a higher-level privilege to the system. Software flaws can be intentional, where the writer has created an exploit which can be used to cause damage. This might include a back-door exploit, where an intruder could connect into a host through some form of network connection, or though a virus or worm. A non-intentional one is where the software has some form of flaw which was unintentional, but which can be used by an intruder. Typical types of non-intentional flaws are: **validation flaws** (where the program does not check for correct input data); **domain flaws** (where data can leak from one program to another); **identification flaws** (where the program does not properly identify the requester); and **logical problems** (where the program does not operate correctly with certain logical steps).

On problem with IDS system is that they cannot investigate encrypted content, which is setup through an encryption tunnel. These tunnels are often used to keep data private when using public networks. It is thus important that the usage of encryption tunnels on corporate network should be carefully used, as threats within them may not be picked-up, and virus/worm scanners and IDS systems will not be able to decrypt the traffic.

2.5 Host/network-based intrusion detection

An intrusion detection system (IDS) can be placed within the network to monitor network traffic, such as looking for known attacks or virus signatures, or can be placed on hosts, where they can detect an actual host intrusion (Figure 2.7). Unfortunately a network-based intrusion detection system cannot obviously decrypt encrypted network data packets, such as with an encryption tunnel (such as such an IPSec connection), thus, in a highly secure network, it is important to run intrusion detection systems on hosts. With encrypted data threats could be hidden from the IDS, as they can be overcome by intruders who know their operation. This is one of the reasons that many organisations do not use IPSec within their systems, and only use it to connect to the perimeter of the network. Some organisations even have net-

work sensors on the network which detect the possible presence of remote connections, and, where possible, the detection of encryption tunnels.

Overall an IDS, just as a firewall, can either be **stateful** or **stateless**. With stateless, the IDS does not have to remember any proceeding data packets, and the state that a connection is in. This will thus have very little overhead as the IDS can discard the packet after it is finished with them. With a stateful IDS, the IDS remembers the previous data packets, and the state of the current connection. This, thus, requires a great deal of memory and buffering, but will be able to understand stateful attacks, and attacks which span over several data packets. For example, if virus was contained with an email, and the email was split into data frames for 1500 bytes, the virus could end up spanning across two data frame, and thus the IDS looking at each data frame at a time would not detect the virus. A stateful IDS, though, can crash if an intruder sends a sequence of data packets into the network, but misses one out, so that the IDS buffers all the other ones, waiting for the missing one, but overruns its buffer size, and crashes.

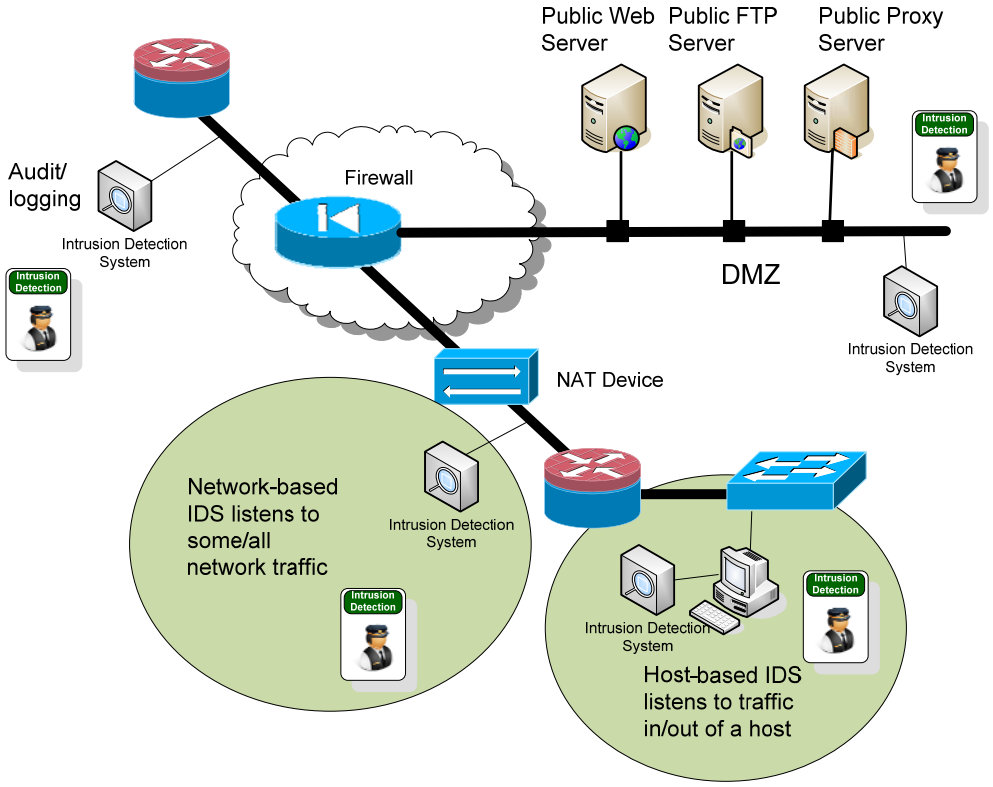


Figure 2.7 Intrusion pattern

There are, though, several ways that an IDS can be tricked in its detection. One is with the creation of a denial-of-service against the IDS, where the network traffic is too great for it to cope with. Another is to stagger the threat over several data packets the IDS must be able to backtrack for connections, and buffer each of the received packets. This obviously has a great effect on its performance, and the more it checks, and backtracks, the slower it is likely to become. As a default, the host-based IDS can

be seen as the last line of defence, where a threat has been able to transverse over the network, and end-up at the host, without being stopped (Figure 2.8).

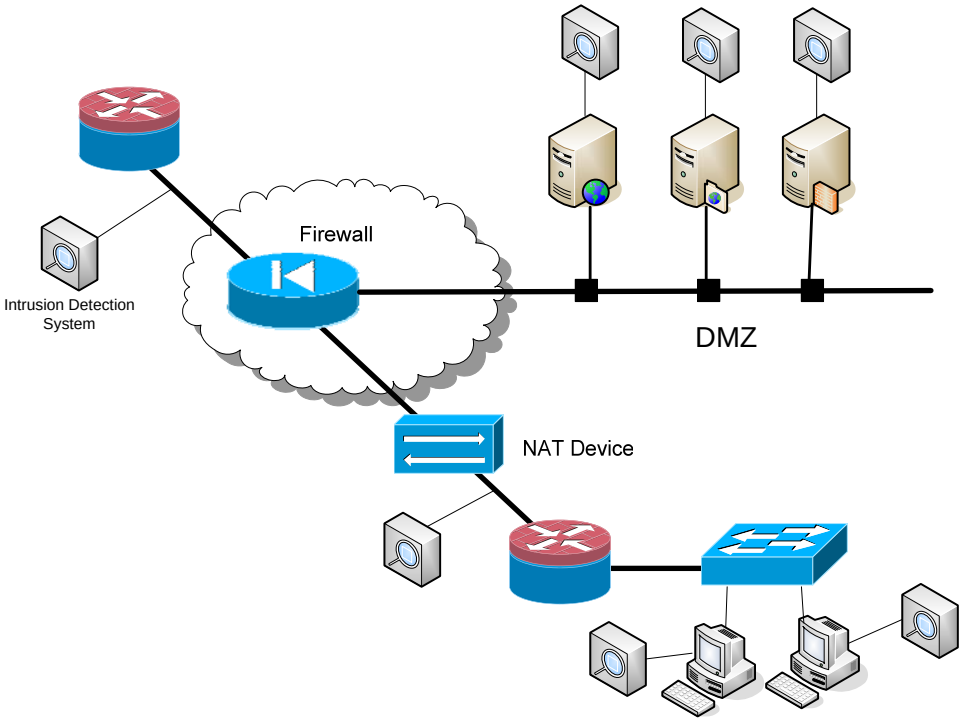


Figure 2.8 IDS

2.6 Placement of the IDS

As an extension of this, the IDSs' can be placed on the servers within the DMZ, and on trusted servers (as illustrated in Figure 2.9). It is also important to place IDS agents on either side of a firewall, as an agent placed on the trusted side of a firewall may not be able to detect an attack which has been blocked, thus agents on either side will detect attacks which have been blocked, and also any that have been allowed to transverse through the firewall. An IDS agent on the untrusted side of the perimeter will thus detect an attack, on the main firewall.

The placement of the IDS on certain devices is important. If it is placed on a hub it can listen to all of the traffic that is on the hub (Figue 2.10). If it is placed on a network switch, it cannot listen to any of the traffic, unless it is configured to forward traffic to a monitoring port. One type of system which can capture data packets from the network is Cisco's SPAN (Switched Port Analyser), which monitors traffic entering the switch (ingress traffic), and traffic leaving the switch (egress traffic). An example of SPAN is shown in Figure 2.11 where the first switch port (FA0/1) monitors FA0/2 and FA0/5, along with the whole of VLAN2. Thus the switch can monitor individual ports, or complete VLANs.

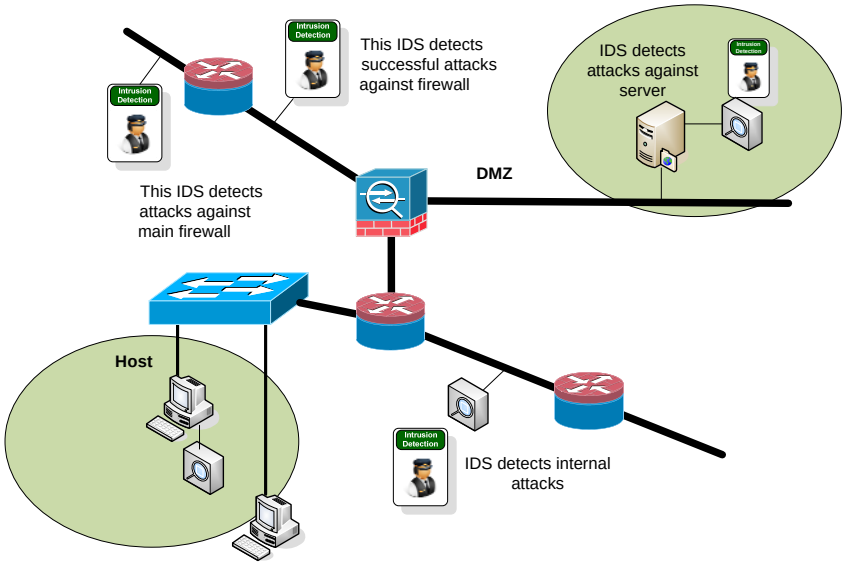


Figure 2.9 Using agents to detect attacks

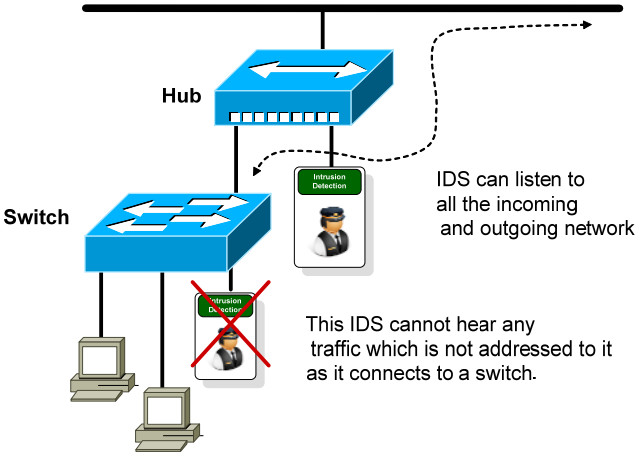
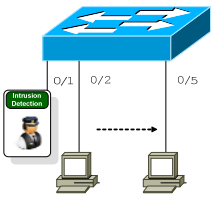


Figure 2.10 Agent detection




```
interface FastEthernet0/1
 port monitor FastEthernet0/2
 port monitor FastEthernet0/5
 port monitor VLAN2
!
interface FastEthernet0/2
!
interface FastEthernet0/3
 switchport access vlan 2
!
interface FastEthernet0/4
 switchport access vlan 2
!
interface FastEthernet0/5
!
interface VLAN1
 ip address 192.168.0.1 255.255.255.0
 no ip directed-broadcast
 no ip route-cache
!
```

Figure 2.11 SPAN detection

2.7 SNORT

The key foundation of most types of data packet detection is in the usage of libpcap (for Unix-based systems) and for Windows-based systems with the **WinPcap** libraries (which have been used in the software tutorials in Chapter 1). Many tools build on these including Snort [1], tcptrace (to identity TCP sessions), tcpflow (to reconstruct TCP sessions) and Ethereal/Wireshark (to capture network traffic). Snort is one of the most widely-used IDS's, and can detect both signature- and anomaly-based detection. In order not to burden the main processes on a machine, often Snort runs as a background process and initially reads-in a set of rules (*filename*.rules) and monitors the network traffic to produce event data and a log (Figure 2.12).

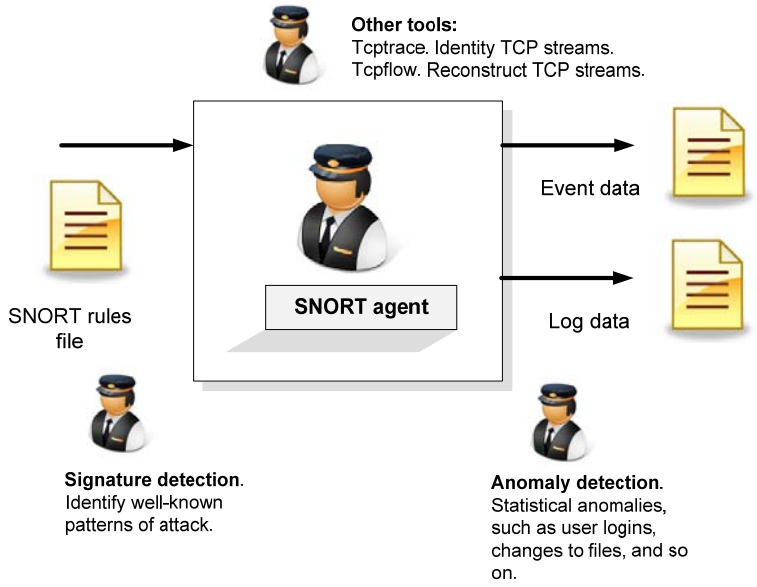


Figure 2.12 Snort

The basic format of a Snort rule header is:

[ACTION] [PROTOCOL] [ADDRESS] [PORT] [DIRECTION] [ADDRESS] [PORT]

which is then followed by options. A basic statement is:

```
alert tcp any any -> 192.168.1.0/24 111
               (content:"|00 01 86 a5|"; msg:"mountd access"; sid:999)
```

where the first word on the rule is the action, such as:

`alert`	Generate an alert and log packet
`log`	Log packet
`pass`	Ignore the packet
`activate`	Alert and activate another rule
`dynamic`	Remain idle until activated by an activate rule

The second part of the rule defines the protocol, such as:

```
tcp       udp
icmp      ip
```

where TCP and UDP are transport layer protocols, while ICMP and IP are Internet (/network) layer protocols. The next few fields define the source and destination of the traffic, as illustrated in Figure 2.13. The source and destination address can be defined as **any** or with an IP address and a subnet mask. For example 192.168.1.0/24 includes a range of addresses from 192.168.1.0 to 192.168.1.255. Along with this, the TCP/UDP port(s) can be defined as either: **any**; a range of ports (*m*:*n* which is port *m* to port *n*); or a specified port. It should be remembered that when a client connects to a server, the client uses its own source port, and it connects, typically, to a well-known port on the server. For example, a client which connects to a Web server, would connect to a destination port of 80, and with a unique source port, such as port 1111. In this case, when the data packets leave the client, the destination port will be 80, and the source port will be 1111. When the data returns from the server, it will have a source port of 80 and a destination one of 1111. It is thus key that the -> is pointing in the correct direction, otherwise the < > can be used for both directions.

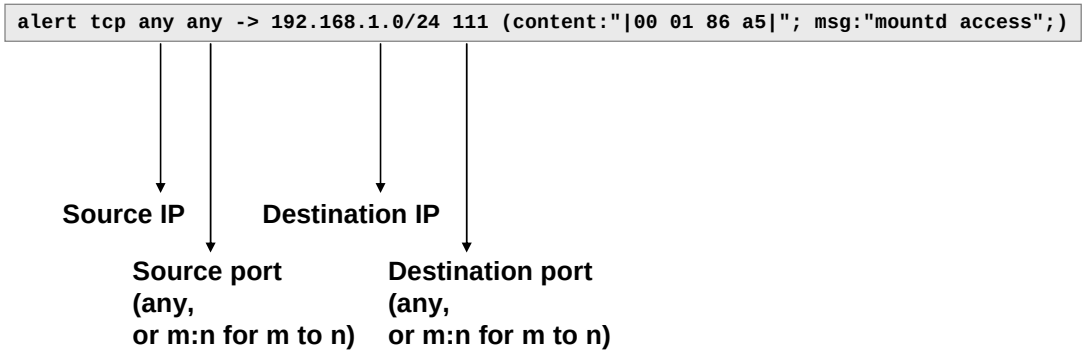


Figure 2.13 Example of source and destination addresses in a Snort rule

Some rules allow a payload in the data packet to be detected. An example of this is given in Figure 2.14. For this the **content** element is used to detect a certain sequence in the data packet. This can be defined either in hexadecimal format (between | and |) or in a plaintext format. Along with this the content element can have several modifiers, such as *offset*, *distance* and *within* which modify the operation of the search. The end part of the rule in Figure 2.14 displays a message if the rule has been activated. There are also various configuration commands that can be used in the rules file, such as:

- config decode_arp (snort –a)
- config payload
- config decode_data_link
- config interface
- config nolog - disable logging, but alerts still occur
- config quiet (snort –q)
- config verbose (snort –v)

- config show_year
- config min_ttl:x

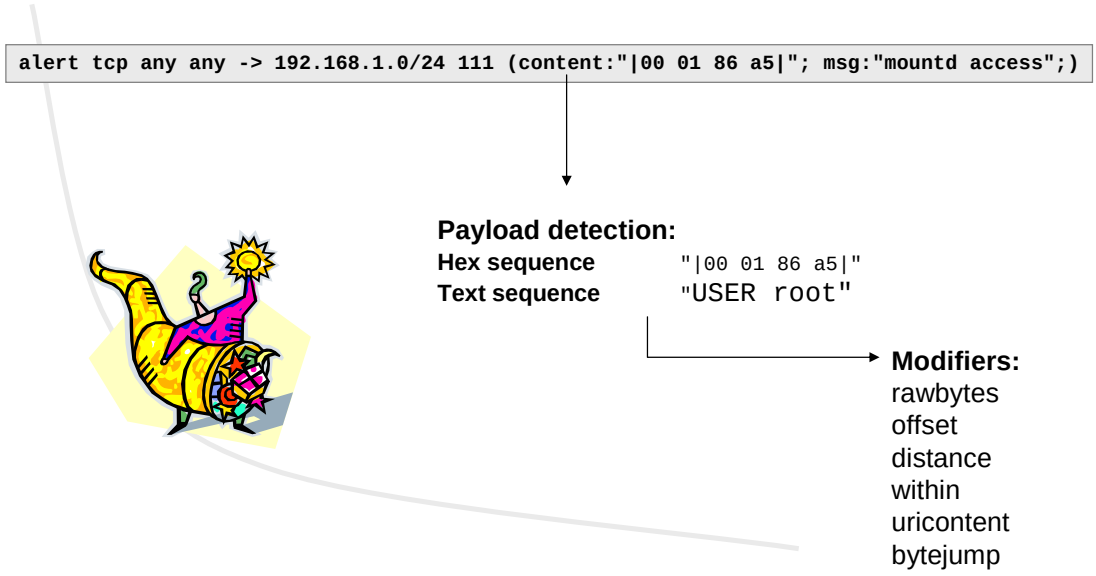


Figure 2.14 Example Snort rule

The SID and REV options are used to give each threat a unique ID, and range from:

- Less 100 Reserved for future use.
- Between 100 and 1,000,000 are rules included with the Snort distribution.
- More than 1,000,000 is for local rules.

For example: sid:336; rev:7; represents an attempt to change to the system administrator's account in FTP.

2.8 Example rules

There are two main variables which are typically defined in the rules files. These are: $HOME_NET which defines all the nodes on our own network; and $EXTERNAL_NET which is every network outside our own network. In the following script the alert is generated when there is a flow of traffic from the external network, on any port, to our own network on port 21 (the FTP port). It then detects that the external intruder is trying to change the current directory to the root's home directory (using the `CWD ~root` command):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
   (msg:"FTP CWD ~root attempt"; flow:to_server,established;
   content:"CWD"; nocase; content:"~root"; nocase;
   distance:1; sid:999)
```

where **nocase** defines that the case of the word is ignored, thus "CWD", and "cwd" would both be detected. The following rule detects incoming traffic which is des-

tined for an FTP server on our own network which tries to get the password file (using the `RETR passwd` FTP command):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
   (msg:"FTP passwd retrieval attempt"; flow:to_server,established;
   content:"RETR"; nocase; content:"passwd"; sid:999)
```

2.8.1 P2P detection

Peer-to-peer (P2P) operations are to be avoided in many organisations, as the laws against them are still being developed. One of the most popular P2P is Kazaa which uses port 1214 to allow a remote peer connection, such as:

```
> netstat -a
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    bills:http             bills:0                LISTENING
  TCP    bills:epmap            bills:0                LISTENING
  TCP    bills:https            bills:0                LISTENING
  TCP    bills:microsoft-ds     bills:0                LISTENING
  TCP    bills:1025             bills:0                LISTENING
  TCP    bills:1214             bills:0                LISTENING
  TCP    bills:2869             bills:0                LISTENING
  TCP    bills:3620             bills:0                LISTENING
  TCP    bills:5679             bills:0                LISTENING
  TCP    bills:1029             bills:0                LISTENING
```

Peer-to-peer program is listening on port 1214

Thus to detect Kazaa activities for a GET message:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 1214
   (msg:"P2P (kazaa/morpheus) GET request";
   flow:to_server,established; content:"GET "; sid:999)
```

2.8.2 MSN Messenger

To detect MSN Messenger, which uses port 1863 for its communications, the MSG command is detected in the data packet payload:

```
alert tcp $HOME_NET any <> $EXTERNAL_NET 1863 (msg:"CHAT MSN
   message"; flow:established; content:"MSG "; sid:999)
```

2.8.3 Virus/Worm detection

For a virus or worm, normally the signature of its propagation is detected, such as:

```
alert tcp any any -> any 139 (msg:"Virus - Possible QAZ Worm";
   flags:A; content: "|71 61 7a 77 73 78 2e 68 73 71|"; sid:999)
```

which detects the worm propagating through port 139, and detects the A flag in the TCP header (A – acknowledge), with a hex pattern of 71 … 71. For a virus, Snort can detect its propagation to an email server (on port 25 - SMTP), such as with a certain file attachment (in this case with a .VBS file attachment):

```
alert tcp $SMTP_SERVERS any -> $EXTERNAL_NET 25
   (msg:"VIRUS OUTBOUND .vbs file attachment";
   flow:to_server,established; content:"Content-Disposition|3a|";
   content:"filename=|22|"; distance:0; within:30;
   content:".vbs|22|"; distance:0; within:30; nocase; sid:999)
```

This detects the name of the file as "*filename*.vbs", where the quotes (") identifies the ASCII character equivalent of 22 in hexadecimal ([2]), and 3a represents a ":".

2.8.4 Sweeps

One activity which typically indicates a potential future security breach is sweeping activities. This typically involves: TCP/UDP sweeps (as illustrated in Figure 2.15); ping sweeps (as illustrated in Figure 2.16), OS identification, and account scans (Figure 2.17).

PORT SCANS. For port sweeps an intruder may scan certain hosts or every host on a subnet, to determine the ports which they have open, as certain ports could be used to gain a foothold on the host. Programs such as **nmap** [3] for example can scan whole networks looking for open ports. A key objective of Snort is to detect this type of activity. Luckily Snort has a pre-processor rule for this, which acts before other rules. An example is:

```
sfportscan: proto  { all } memcap { 10000000 } sense_level { low }
```

where the arguments might include:

- **proto**. This can be tcp, udp, icmp, ip or all, and are the types of protocol scans to be detected.
- **scan_type**. This can be portscan, portsweep, decoy_portscan, distributed_portscan or all, and defines the scan type to be detected.
- **sense_level**. This can be low, medium or high, and defines the sensitity of the portscans. A low sense level detects response errors, such as ICMP unreachables. Medium sensitivity level detects portscans and filtered portscans (which are portscans that do not have any responses). High sensitivity level has a lower threshold than medium and has a longer time window to detect sweeps.
- **Memcap**. This defines the maximum memory size (in bytes) – this limits the possibility of buffer overflows.
- **Watch_Ip**. This defines the hosts that are to be detected.

To save to a file named portscan.log (scan.rule):

```
preprocessor flow: stats_interval 0 hash 2
preprocessor sfportscan: proto { all } scan_type { all }
              sense_level { low } logfile { portscan.log }
```

It is always important to understand the ports that are open on a computer, such as with running NMAP:

```
C:\> snort -c scan.rule -dev -i 3 -p -l c:\\bill -K ascii
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file scan.rule
,-----------[Flow Config]---------------------
| Stats Interval:  0
| Hash Method:     2
| Memcap:          10485760
| Rows  :          4096
| Overhead Bytes:  16388(%0.16)
`-------------------------------------------
Portscan Detection Config:
    Detect Protocols:  TCP UDP ICMP IP
    Detect Scan Type:  portscan portsweep decoy_portscan distributed_portscan
    Sensitivity Level: Low
    Memcap (in bytes): 1048576
    Number of Nodes:   3869
    Logfile:           c:\\bill/portscan.log

Tagged Packet Limit: 256
```

Then for a scan:

```
C:\> nmap -o -A 192.168.0.1
Starting Nmap 4.20 ( http://insecure.org ) at 2007-01-09 21:58 GMT Standard Time
Interesting ports on 192.168.0.1:
Not shown: 1695 closed ports
PORT     STATE SERVICE
80/tcp   open  http
8888/tcp open  sun-answerbook
MAC Address: 00:0B:44:F5:33:D5 (The Linksys Group)
Nmap finished: 1 IP address (1 host up) scanned in 1.500 seconds
```

The resulting log then gives the trace of the port sweep and scan:

```
Time: 08/17-14:41:54.495296
event_ref: 0
192.168.0.3 -> 63.13.134.49 (portscan) TCP Portsweep
Priority Count: 5
Connection Count: 135
IP Count: 43
Scanned IP Range: 63.13.134.49:216.239.59.99
Port/Proto Count: 1
Port/Proto Range: 80:80

Time: 08/17-14:42:52.431092
event_ref: 0
192.168.0.3 -> 192.168.0.1 (portscan) TCP Portsweep
Priority Count: 5
Connection Count: 10
IP Count: 5
Scanned IP Range: 66.249.93.165:192.168.0.7
Port/Proto Count: 3
Port/Proto Range: 80:2869

Time: 08/17-14:42:52.434852
event_ref: 0
192.168.0.3 -> 192.168.0.1 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 9
IP Count: 1
Scanner IP Range: 192.168.0.3:192.168.0.3
Port/Proto Count: 10
Port/Proto Range: 21:636
```

PING SCANS. With ping scans, the intruder tries to determine the hosts which are active on a network. An example of detecting a Window's ping sweep is:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (
   msg:"ICMP PING Windows"; itype:8; content:"abcdefghijklmnop";
   depth:16; sid:999)
```

where an ICMP ping packet is detected with the standard contents of "abc….op". An example of the contents of a ping request is:

```
0000  00 0c 41 f5 23 d5 00 15  00 34 02 f0 08 00 45 00   ..A.#... .4....E.
0010  00 3c 10 7c 00 00 80 01  a6 8f c0 a8 01 64 c0 a8   .<.|.... .....d..
0020  01 01 08 00 60 55 04 00  e9 06 61 62 63 64 65 66   ....`U.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                    wabcdefg hi
```

And a ping reply:

```
0000  00 15 00 34 02 f0 00 0c  41 f5 23 d5 08 00 45 00   ...4.... A.#...E.
0010  00 3c 10 7c 00 00 96 01  90 8f c0 a8 01 01 c0 a8   .<.|.... ........
0020  01 64 00 00 68 55 04 00  e9 06 61 62 63 64 65 66   .d..hU.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                    wabcdefg hi
```

OS SCANS. For OS identification the intruder searches hosts for certain machines, which possibly have an OS weakness, such as searching for Windows 95 machines, as these tend to have FAT32 file systems which have very little security associated with them. For account scans, an intruder may scan the user ID's for weak passwords. where the tests are:

- **TSeq**. This is where SYN packets are sent, and the TCP sequence numbers are analysed.
- **T1**. This is a SYN packet with certain options (WNMTE) set is sent to an open TCP port.
- **T2**. This is a NULL packet with options (WNMTE) and is sent to an open TCP port.
- **T3**. This is a SYN,FIN,PSH,URG packet with options (WNMTE), and sent to an open TCP port.
- **T4**. This is an ACK packet with options (WNMTE) and is sent to an open TCP port.
- **T5**. This is a SYN packet with options (WNMTE) and is sent to a closed TCP port.
- T6. This is an ACK packet with options (WNMTE) and is sent to a closed TCP port.
- **T7**. This is a FIN,PSH,URG packet with options (WNMTE) and is sent to a closed TCP port.
- **PU**. This is a packet sent to a closed UDP port.

For example the following is a fingerprint from XP Professional:

```
TSeq(Class=RI%gcd=<8%SI=<2959A&>356%IPID=I)
```

```
T1(DF=Y%W=FAF0|402E%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

where:

- **Resp**: defines whether the host responds. Y - for a response, and N - no response.
- **DF:** defines whether the host responds with a "Don't Fragment" bit set in response. Y - DF was set, N - DF was not set.
- **W:** defines the acknowledgement sequence number response and is the Window advertisement size sent by the host. ACK 0 - ack zero, S - ack sequence number, S++ - ack sequence number + 1.
- **Flags:** this defines the flags set in response. S = SYN, A = ACK, R = RST, F = FIN, U = URG, P = PSH.
- **Ops:** this is the options set for the response. M - MSS, E - Echoed MSS, W - Window Scale, T - Timestamp, and N - No Option.

For example `DF=Y%W=FAF0|402E%ACK=S++%Flags=AS%Ops=MNWNNT`

defines that the "Don't Fragment" bit is set, the Window size is set to FAF0 or 402E, the acknowledgement sequence number is set to one more than the requesting packet, the flags set to ACK/SYN, with Options of MNWNNT.

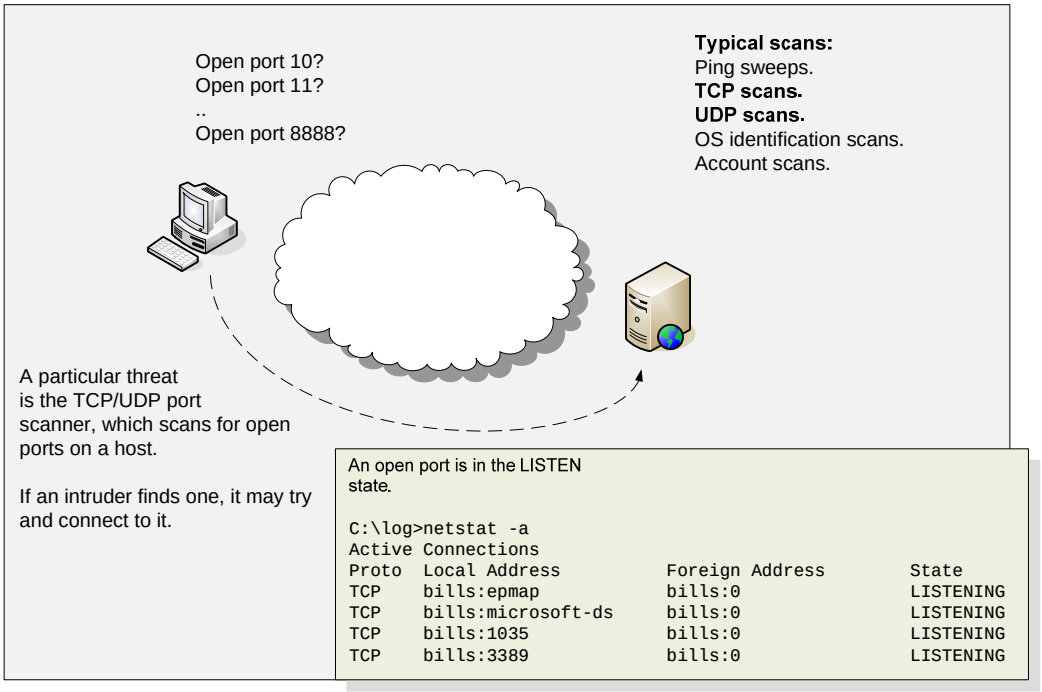


Figure 2.15 TCP/UDP port sweeps

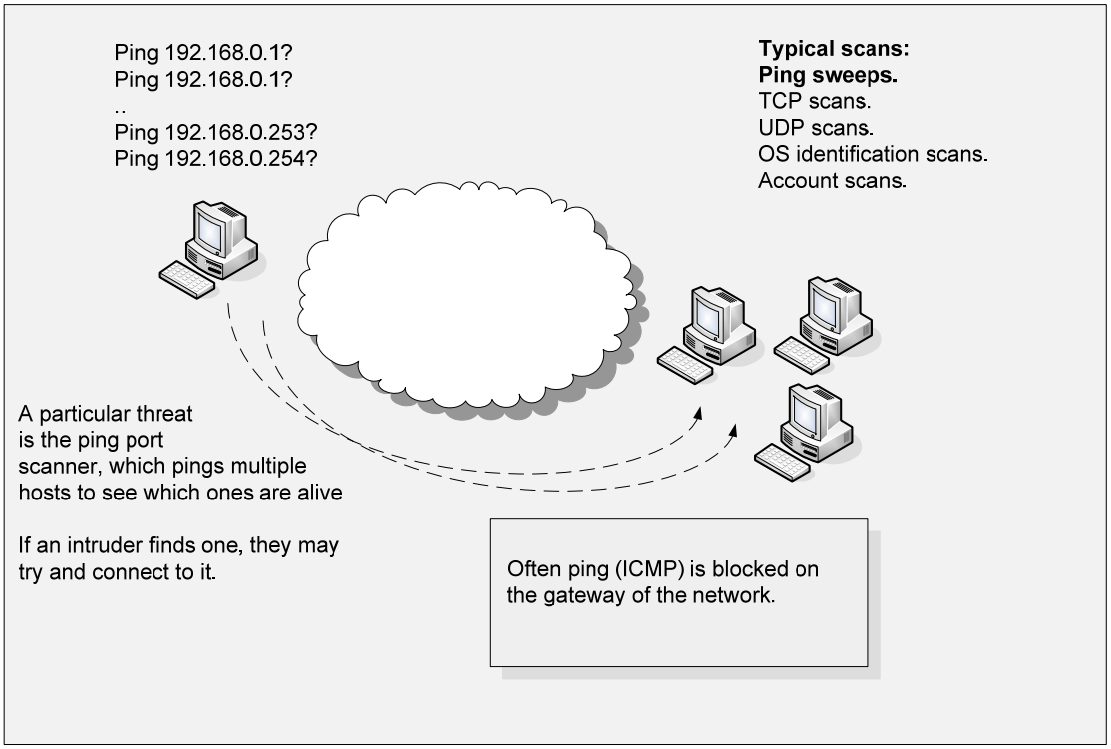


Figure 2.16 Ping sweeps

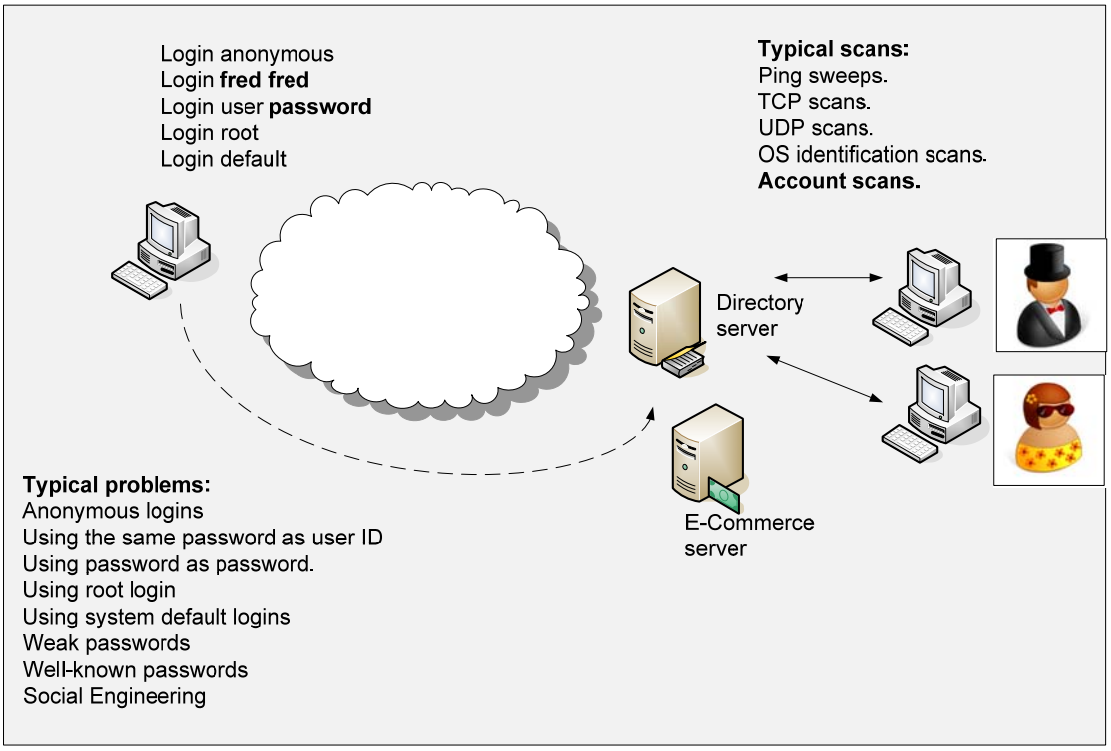


Figure 2.17 Account sweeps

2.9 Running Snort

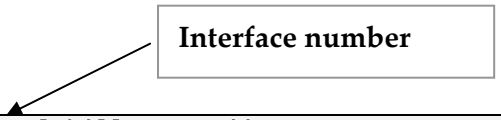
A typical method of running Snort is:

```
snort -v -c bill.rules -dev -i 1 -p -l c:\\bill -K ascii
```

where -c identifies the rules file, -v identifies verbose mode, -l defines the directory for the alerts file (alert.log), and the –K option defines that the format of the log. An example rule in the file is:

```
alert tcp any any -> any any (content:"the"; sid: 999;
   msg:"The found ....";)
```

A run of the Snort gives:




```
C:\Snort\bin> snort -v -c bill.rules -dev -i 1 -l bill -K ascii
Running in IDS mode
        --== Initializing Snort ==--
Initializing Output Plugins!
Var '_ADDRESS' redefined
Var '\Device\NPF_{3DFE7A22-72FF-458C-80E2-C338584F5F71}_ADDRESS' defined, value
len = 25 chars, value = 192.168.0.0/255.255.255.0
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file bill.rules
+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
1 Snort rules read...
1 Option Chains linked into 1 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++

Tagged Packet Limit: 256

+----------------------[thresholding-config]----------------------------------
| memory-cap : 1048576 bytes
+----------------------[thresholding-global]----------------------------------
| none
+----------------------[thresholding-local]-----------------------------------
| none
+----------------------[suppression]------------------------------------------
| none
-------------------------------------------------------------------------------
Rule application order: ->activation->dynamic->pass->drop->alert->log
Log directory = bill
Verifying Preprocessor Configurations!
0 out of 512 flowbits in use.

Initializing Network Interface \Device\NPF_{3DFE7A22-72FF-458C-80E2-C338584F5F71
}
Decoding Ethernet on interface \Device\NPF_{3DFE7A22-72FF-458C-80E2-C338584F5F71
}
+--[Pattern Matcher:Aho-Corasick Summary]----------------------
| Alphabet Size    : 256 Chars
| Sizeof State     : 2 bytes
| Storage Format   : Full
| Num States       : 4
| Num Transitions  : 6
| State Density    : 0.6%
| Finite Automatum : DFA
| Memory           : 2.23Kbytes
+-------------------------------------------------------------
        --== Initialization Complete ==--
   ,,_      -*> Snort! <*-
  o"  )~    Version 2.6.1.2-ODBC-MySQL-FlexRESP-WIN32 (Build 34)
   ''''     By Martin Roesch & The Snort Team: http://www.snort.org/team.html
            (C) Copyright 1998-2006 Sourcefire Inc., et al.

Not Using PCAP_FRAMES
01/10-17:52:40.507621 0:15:0:34:2:F0 -> 0:18:4D:B0:D6:8C type:0x800 len:0x86
192.168.0.3:10603 -> 146.176.222.183:2304 TCP TTL:128 TOS:0x0 ID:18857 IpLen:20
DgmLen:120 DF
***AP**F Seq: 0x34EF0A67  Ack: 0x301650BD  Win: 0x40B0  TcpLen: 20
```

```
52 50 59 20 30 20 30 20 2E 20 30 20 35 39 0D 0A  RPY 0 0 . 0 59..
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 70  Content-Type: ap
70 6C 69 63 61 74 69 6F 6E 2F 62 65 65 70 2B 78  plication/beep+x
6D 6C 0D 0A 0D 0A 3C 67 72 65 65 74 69 6E 67 3E  ml....<greeting>
3C 2F 67 72 65 65 74 69 6E 67 3E 45 4E 44 0D 0A  </greeting>END..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

After generating some network traffic with the word “The” in it gives:

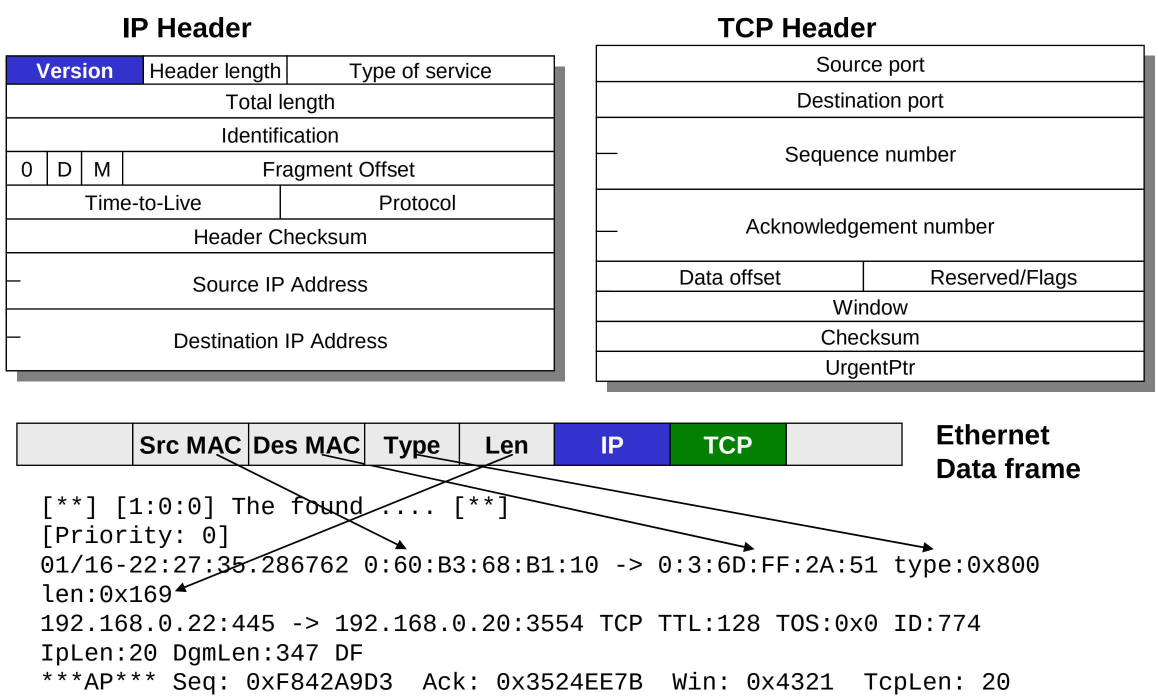
```
[**] [1:999:0] The found .... [**]
[Priority: 0]
01/10-17:59:05.921463 0:15:0:34:2:F0 -> 0:18:4D:B0:D6:8C type:0x800 len:0x229
192.168.0.3:10688 -> 66.249.93.99:80 TCP TTL:128 TOS:0x0 ID:19086 IpLen:20
DgmLen:539 DF
***AP*** Seq: 0x54455E9E  Ack: 0x25828CFC  Win: 0x4308  TcpLen: 20

[**] [1:999:0] The found .... [**]
[Priority: 0]
01/10-17:59:13.124776 0:15:0:34:2:F0 -> 0:18:4D:B0:D6:8C type:0x800 len:0x264
192.168.0.3:10688 -> 66.249.93.99:80 TCP TTL:128 TOS:0x0 ID:19096 IpLen:20
DgmLen:598 DF
***AP*** Seq: 0x54456091  Ack: 0x2582A1E9  Win: 0x3F4D  TcpLen: 20

[**] [1:999:0] The found .... [**]
[Priority: 0]
01/10-17:59:13.236763 0:15:0:34:2:F0 -> 0:18:4D:B0:D6:8C type:0x800 len:0x1BB
192.168.0.3:10690 -> 143.252.148.160:80 TCP TTL:128 TOS:0x0 ID:19102 IpLen:20
DgmLen:429 DF
***AP*** Seq: 0xCA6A04D7  Ack: 0xB92991CD  Win: 0x4470  TcpLen: 20
```

Alert!

it can be seen that the date and time is logged for each alert. The source and destination MAC addresses are also defined (Figure 2.18), along with the IP (Figure 2.19) and TCP parts (as shown in Figure 2.20).



Remember… IP addresses can be spoofed … the MAC addresses can’t (well it’s difficult).

Figure 2.18 Ethernet headers

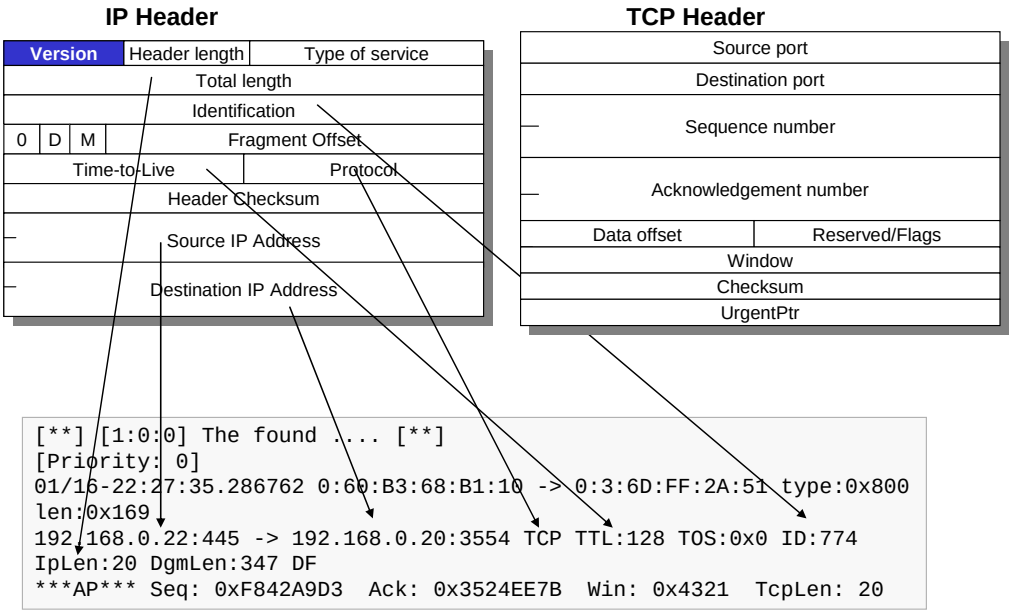


Figure 2.19 IP headers

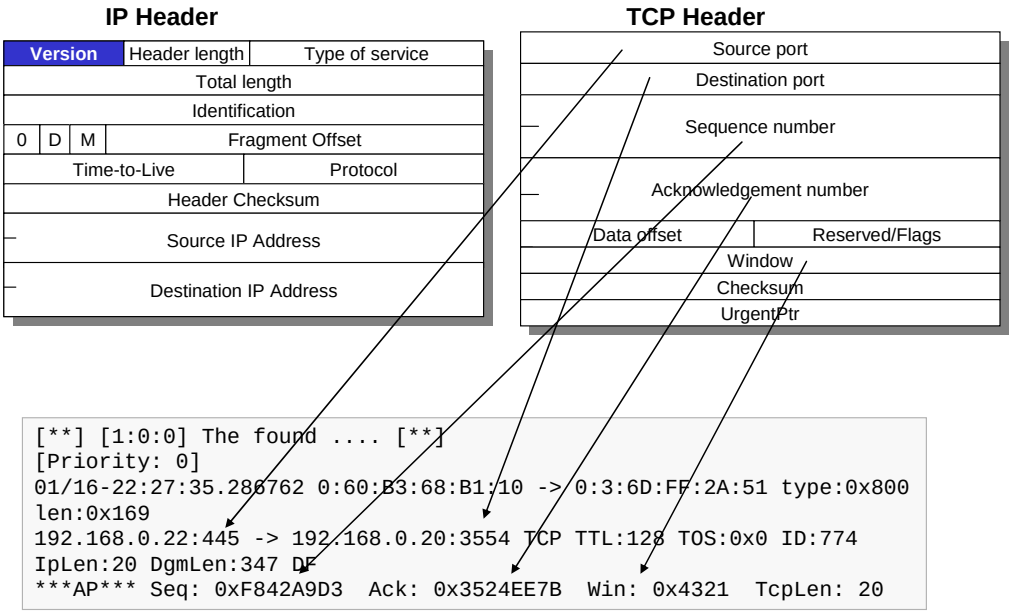


Figure 2.20 TCP headers

Snort creates logs for each port-to-port connection for each host on the network (as illustrated in Figure 2.21). A key element of detecting different types of traffic is in the analysis of the TCP flags. These are (UAPRSF):

- U is the urgent flag (URG).
- A the acknowledgement flag (ACK).
- P the push function (PSH).
- R the reset flag (RST).
- S the sequence synchronize flag (SYN).
- F the end-of-transmission flag (FIN).

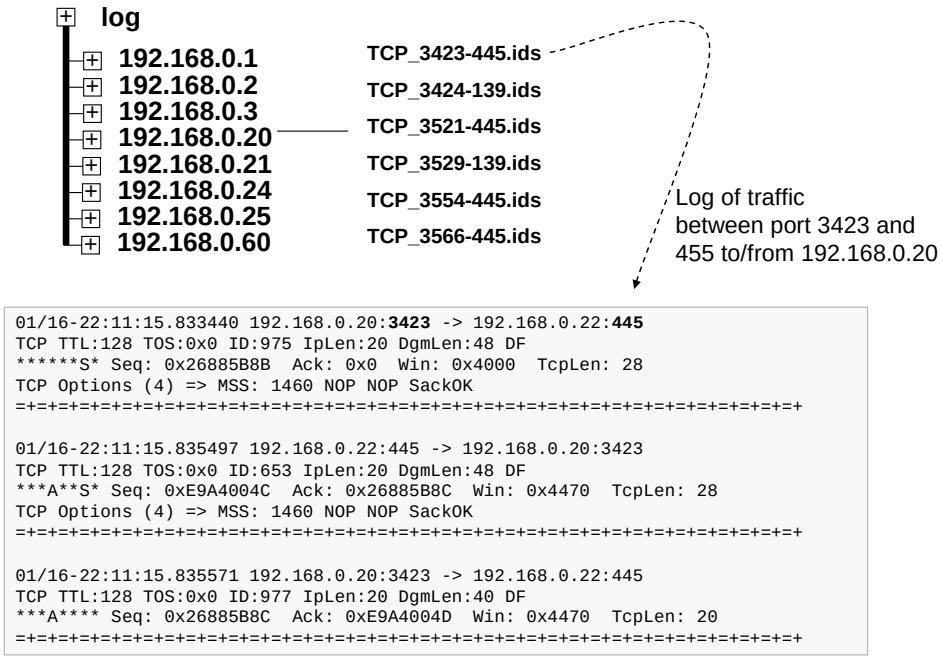


Figure 2.21 TCP headers

2.9.1 TCP flags

A client-server connection normally involves an initial handshaking to negotiate the connection, such as:

```
     Client                                                    Server
  1.  CLOSED                                                   LISTEN
  2.  SYN-SENT  -> <SEQ=999><CTL=SYN>                          SYN-RECEIVED
  3.  ESTABLISHED    <SEQ=100><ACK=1000><CTL=SYN,ACK>    <-    SYN-RECEIVED
  4.  ESTABLISHED -> <SEQ=1000><ACK=101>   <CTL=ACK>          ESTABLISHED
```

This is known as the **three-way handshake**. It involves: a <**SYN>** from the client to the server; a <SYN, ACK> from the server to the client (to acknowledge the connection from the server); and an <ACK> from the client to the server (to finalise the connection). The **SYN** flag is thus important in detecting a client connecting to a server. Thus, an incoming SYN flag is important in detecting the start of a connection from outside the network to a server inside the network, whereas an outgoing SYN flag identifies a connection to a server outside the network. The main flags are:

F FIN **S** SYN **R** RST **P** PSH
A ACK **U** URG

and the following modifiers can be set to change the match criteria:

+ match on the specified bits, plus any others
***** match if any of the specified bits are set
! match if the specified bits are not set

An example to test for SYN flag is:

```
alert tcp any any -> any any (flags:S; sid:999)
```

It is often important to know the flow direction (such as coming from or going to a sever), the main flow rules options are:

- to_client. Used for server responses to client.
- to_server Used for client requests to server.
- from_client. Used on client responses.
- from_server. Used on server responses.
- established. Established TCP connections.

For example to test for an FTP connection to the user's computer:

```
alert tcp any any -> $HOME_NET 21 (flow: from_client;
   content:"CWD"; nocase; message: "CWD incoming"; sid:999)
```

Figure 2.22 shows an example of the flags that are set. In this case, the A and S flags identify the SYN, ACK sequence (which occurs when the server responds back to the client for the connection. Notice that it goes **S** (SYN), **S/A** (SYN-ACK), and then **A** (ACK), which completes the creation of the client-server connection.

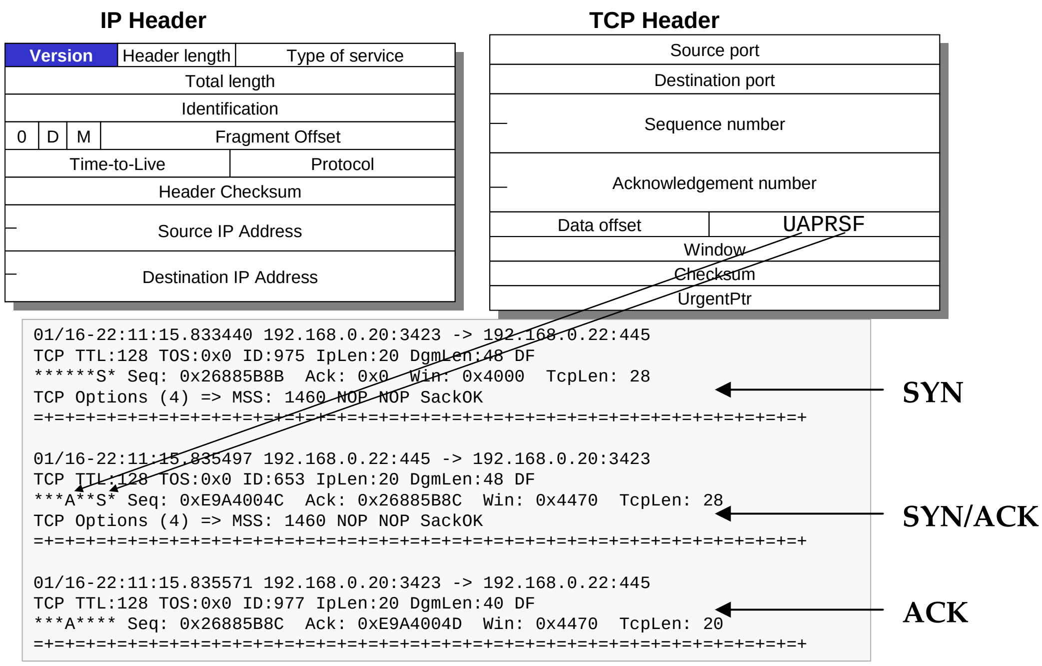


Figure 2.22 Example flags

Along with this Snort can be used to analyse the ARP translations on the network. This gives a pointer to the devices which are asking to resolve the MAC address of a host (Figure 2.23). Note that the ARP requests are sent to everyone on the same network (which is a domain bounded by router ports).

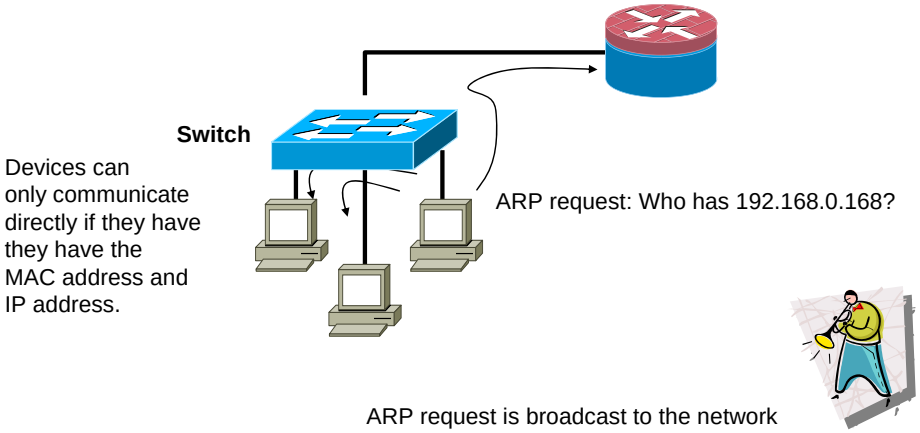


Figure 2.23 ARP log

2.10 User, machine and network profiling

One of the best ways of detecting human behaviour, especially in detecting fraud, is user profiling. For this, an agent can detect a given user, and build up a profile on them (Figure 2.24). If the behaviour of the user changes, it may be that an intruder has used their account. For example a user might type at a speed of 30 words per minute, whereas an intruder who has logged on as the user might be detected if they type at 60 words per minute. This method, though, has many ethical issues, which would have to be overcome before it is implemented in a system.

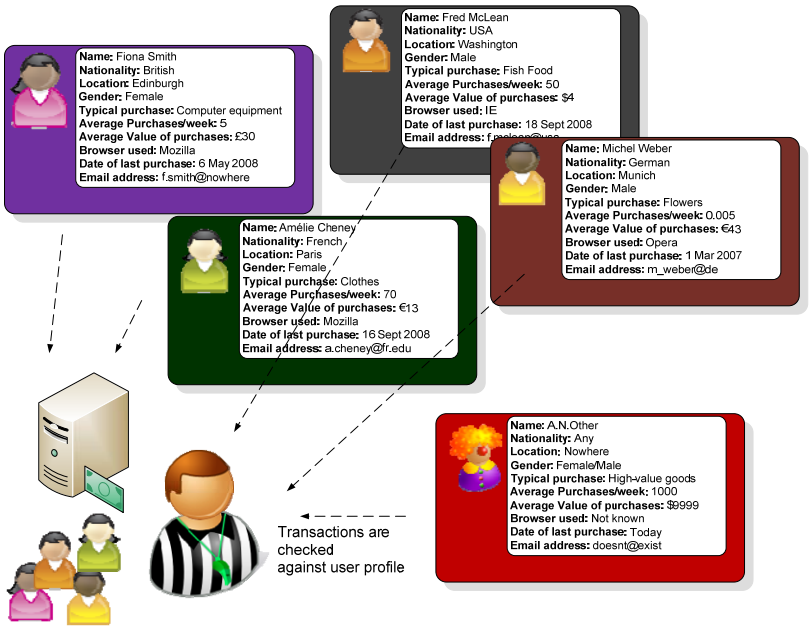


Figure 2.24 User profiling

Typical methods of profiling users might relate to typing speeds, applications which they typically run, common typing errors, working hours, and so on. For host profiling it is possible to define a normal benchmark for a host. For example, a test could be run for one day, and it would profile the machine as:

Processes running range =	20 – 30
CPU utilization (average per minute) =	0 – 30%
Free disk space (average per minute) =	100MB – 1GB
Memory Available =	1.2GB – 2.4GB

Thus if the number of processes increased to 40, then this could be flagged as a deviation from the norm. The calibration and training period is obviously important, in order to not overload the adminstrator with false alerts.
For network profiling, it is possible to listen to network traffic for a given amount of time, and define benchmarks on normal traffic. For example a profile might be:

IP traffic (per hour) =	30-85%
TCP traffic (per hour) =	25-75%
HTTP traffic (per hour) =	30-50%
FTP traffic (per hour) =	0-5%

Thus the detection could be based on monitoring the amount of traffic over hourly periods, and if it went outwith these limits, the system would generate an alert. An example might be if the FTP traffic increased to 10% over an hourly period. This might help identify large amount of uploads/downloads for file transfer.

2.11 Honey pots

Sometimes it is possible to create a honey-pot, which attracts an intruder so that they can be caught before they do any damage. It also can help to identify the propagation of viruses and/or worms. An example of a low interaction honeypot is Honeyd, which uses typically scripts to simulate a host (Figure 2.25). Honey pots are currently under investigation by many researchers, but may have some moral issues, as they can be setup to trap intruders. A honey pot is typically setup with required weaknesses, such as (Figure 2.26):

- Default administrator/password.
- Dummy users with weak passwords.
- Ports open for connection.
- Reacting to virus/worm systems (but simulate conditions).

The main types of honeypots are:

- **High-interaction honeypot**. This simulates all the aspects of the operating system and the device.

- **Low-interaction honeypot**. This simulates only part of the network stack (such as for Honeyd). It can be virtual (from a virtual machine) or simulated by a real machine.

An example script for Honeyd in order to simulate a Windows XP host, which has open ports of 110 (POP-3), 80 (Web), 21 (FTP) and 22 (SSH), and blocked ports of 25 (SMTP) and 139 (NetBIOS):

```
create default
set default personality "Windows XP"
set default default tcp action reset
add default tcp port 110 "sh scripts/pop.sh"
add default tcp port 80 "perl scripts/iis-0.95/main.pl"
add default tcp port 25 block
add default tcp port 21 "sh scripts/ftp.sh"
add default tcp port 22 proxy $ipsrc:22
add default udp port 139 drop
set default uptime 3284460
```

which is using and an example of a simulation of a Cisco PIX firewall with an open Telnet port:

```
### Cisco router
create router
set router personality "Cisco PIX Firewall (PixOS 5.2 - 6.1)"
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router default tcp action reset
set router uid 32767 gid 32767
set router uptime 1327650
# Bind specific templates to specific IP address
# If not bound, default to Windows template
bind 192.168.1.150 router
```

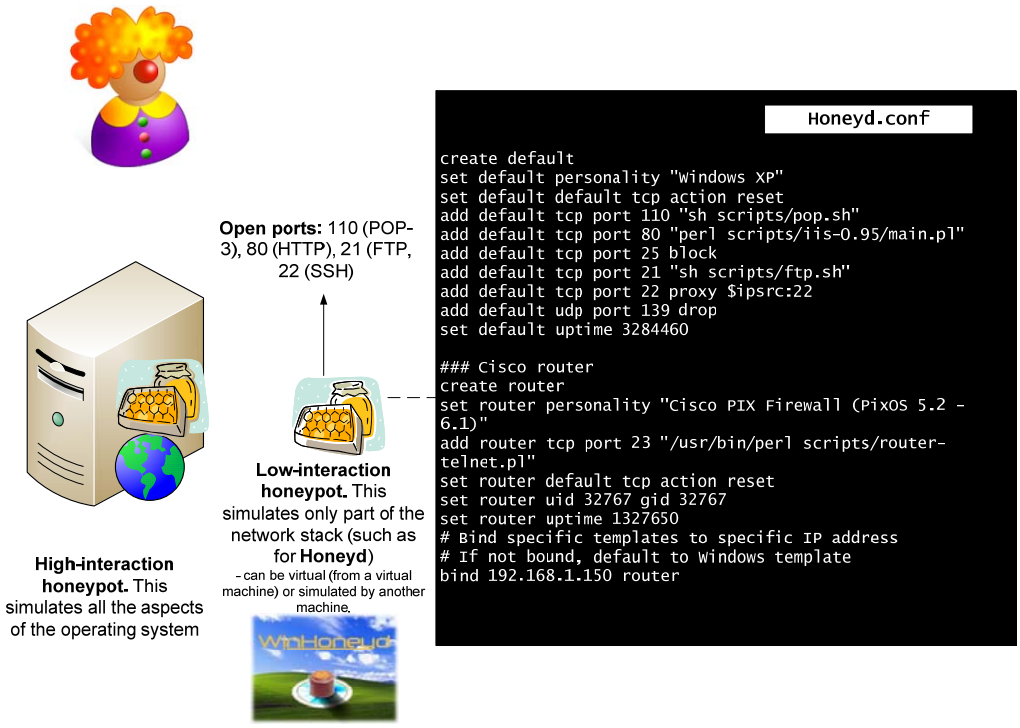


Figure 2.25 Honeyd

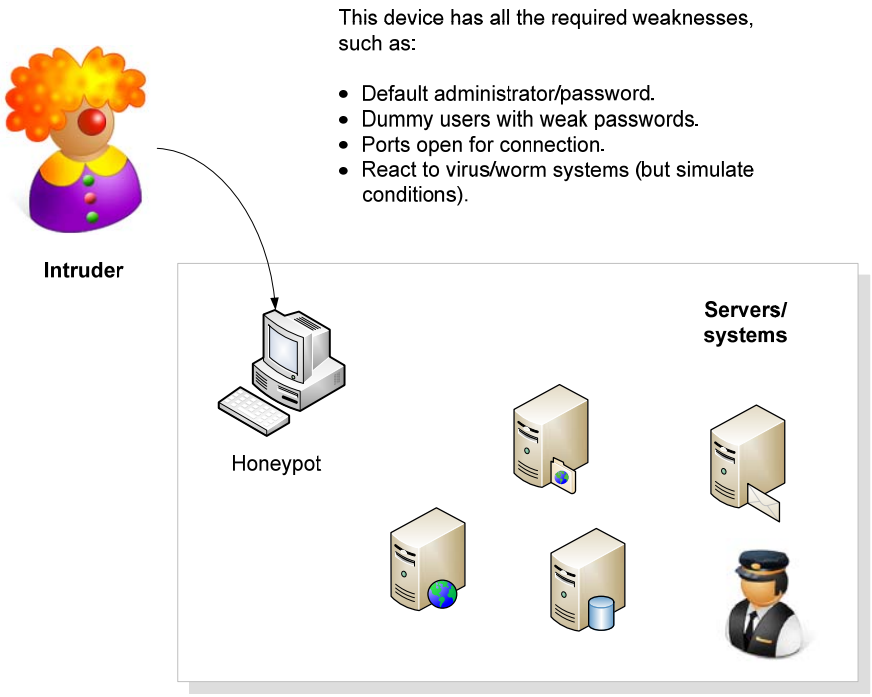


Figure 2.26 Honeypots

2.12 In-line and out-of-line IDSs

Snort is seen as an out-of-line IDS, as it typically passively monitors the data packets and does not take any action. This is defined as an out-of-line IDS (Figure 2.27). An in-line IDS, such as the Cisco IDS is embedded into the Cisco IOS, and can be used to take action on intrusions. In a Cisco IDS, each type intrusion has a unique ID, such as 3041 which relates to a data packet with the SYN and FIN flags set. The main classifications for Cisco IDS signatures are: Information (atomic), Information (compound), Attack (atomic), Attack (compound), where an atomic element identifies one instance of the intrusion, and a compound element identifies more than one intrusion element. An example from a Cisco IDS is:

```
(config)# ip audit ?
  attack     Specify default action for attack signatures
  info       Specify default action for informational signatures
  name       Specify an IDS audit rule
  notify     Specify the notification mechanisms (nr-director or log) for the
             alarms
  po         Specify nr-director's PostOffice information (for sending events
             to the nr-directors
  signature  Add a policy to a signature
  smtp       Specify SMTP Mail spam threshold
(config)# ip audit notify ?
  log          Send events as syslog messages
  nr-director  Send events to the nr-director
(config)# ip audit notify log
(config)# logging 132.191.125.3
(config)# ip audit ?
  attack     Specify default action for attack signatures
  info       Specify default action for informational signatures
  name       Specify an IDS audit rule
  notify     Specify the notification mechanisms (nr-director or log) for the
             alarms
  po         Specify nr-director's PostOffice information (for sending events
             to the nr-directors
  signature  Add a policy to a signature
```

```
  smtp       Specify SMTP Mail spam threshold
(config)# ip audit info ?
  action  Specify the actions
(config)# ip audit info action ?
  alarm  Generate events for matching signatures
  drop   Drop packets matching signatures
  reset  Reset the connection (if applicable)
(config)# ip audit info action drop
(config)# ip audit attack action reset
(config)# ip audit signature ?
  <1-65535>  Signature to be configured
(config)# ip audit signature 1005 disable
(config)# ip audit smtp ?
  spam  Specify the threshold for spam signature
  <cr>
(config)# ip audit smtp spam ?
  <1-65535>  Threshold of correspondents to trigger alarm
(config)# ip audit smtp spam 4
```

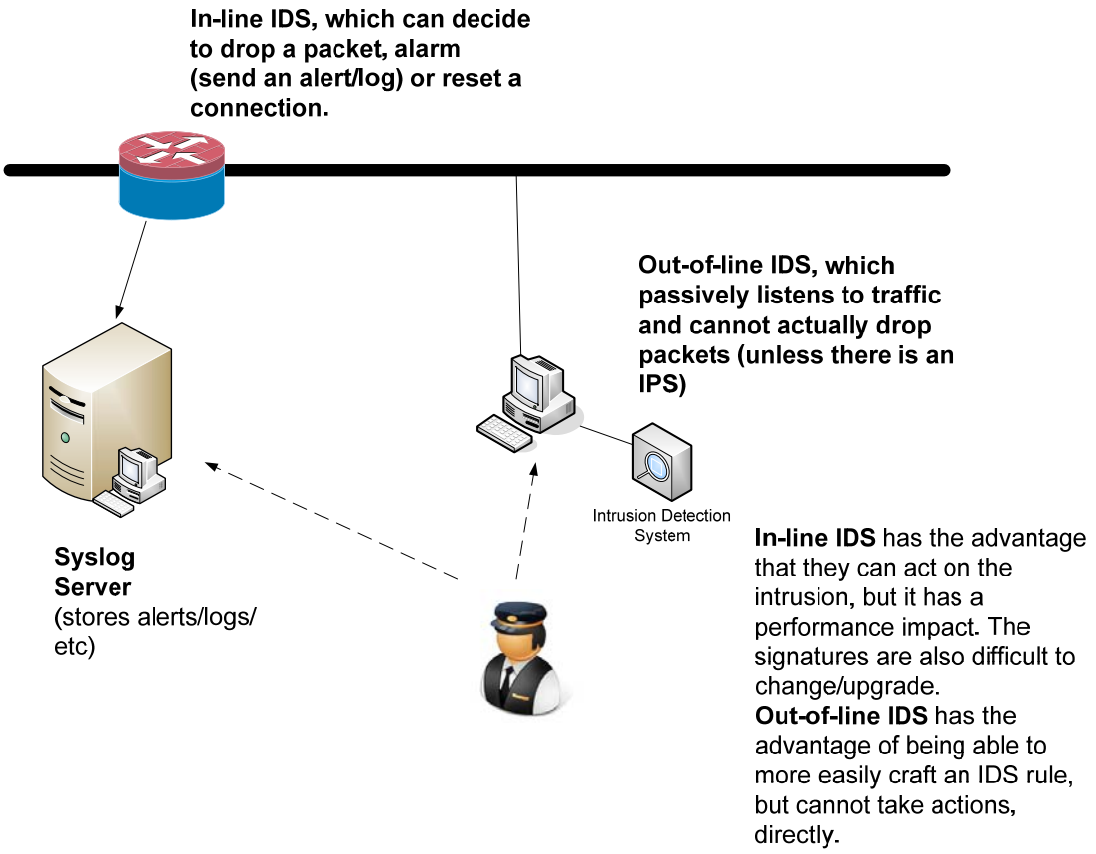


Figure 2.27 IDS (in-line and out-of-line)

2.13 False and true

A key factor in any intrusion detection system is its success in actually determining threats. For this, there are a number of key metrics which defines the success of the system:

- **False positives**. This is the number of intrusions that the IDS failed to spot.
- **False negatives**. This is the number of alerts that were generated that were not actually intrusions, and could thus be wasteful in investigation time.
- **True positives**. This is the number of actual number of intrusions which were correctly identified.

A good IDS will give a high number of true positives against false negatives, as too many false negatives will often cause the administrator to become desensitized to alerts. A key factor in this is often to have some sort of filtering on the alerts, so that key alerts overrule lesser alerts. Also, if the number of false positives is too high compared with the number of true positives, the administrator might feel that the system is missing too main intrusions. There should thus be a continual refinement of the IDS rules in order to give the system the correct balance. Often what happens is that experience of system operations shows the right sensitivity of the system.

2.14 Customized Agent-based IDS

The usage of standard IDSs such as Snort is an excellent method of detecting intrusions, but often they are generalized in their detection engine, and have a significant overhead in detecting certain types of intrusions. It has been shown by many researchers that Snort can be made to miss alerts and even crash on relatively low data throughputs. Thus, in several applications, the use of customized agents are required which focus on detecting certain types of network traffic. Along with this, it can integrate with other system detection elements on a host, such as detecting changes to system files, and in detecting CPU usage. Thus agent-based systems using WinPcap are useful in optimizing intrusions, without the footprint of a full-blown system. The software developed in Section 2.15 focuses on customized agent-based IDS. This system is illustrated in Figure 2.28, where a configuration agent writes the Snort rules, and then invokes the Snort agent, which reads the rule file. The security agent then reads the alerts from Snort.

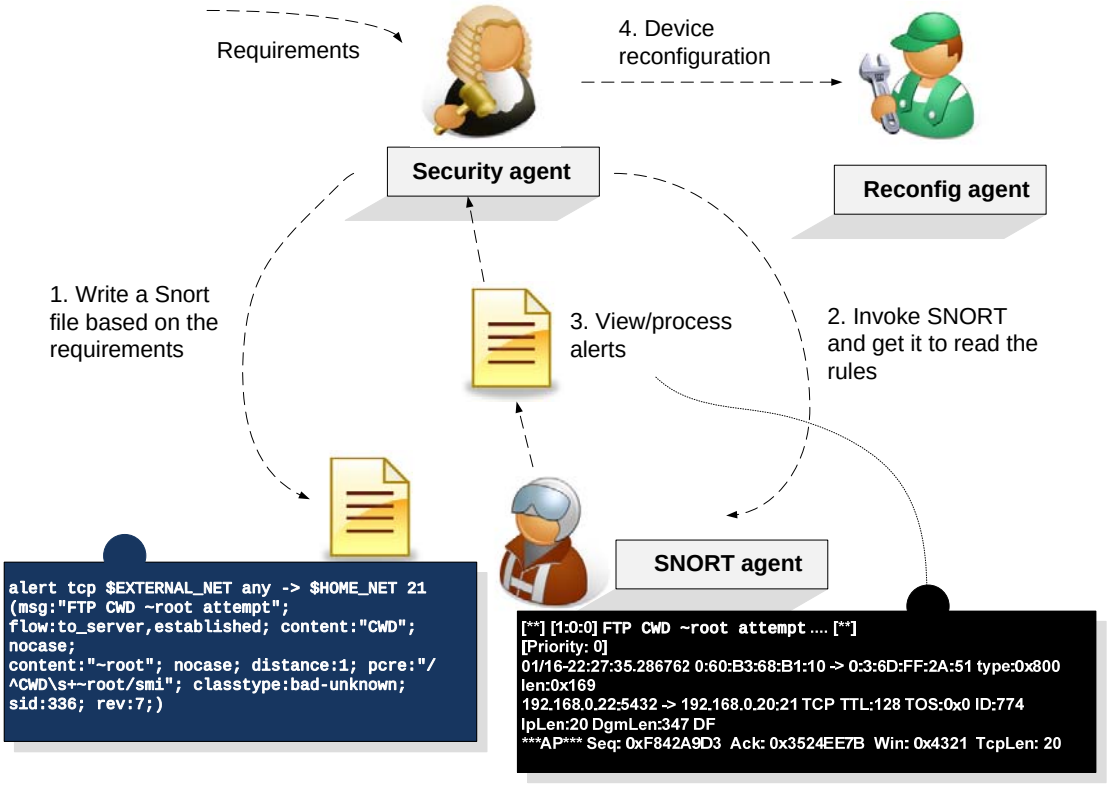


Figure 2.28 Agent-based IDS

2.15 Tutorial

2.14.1 The on-line exercise for this chapter are at:

http://buchananweb.co.uk/security00.aspx

and select **Introduction to IDS [Test]**.

2.14.2 Which Snort command will filter for outgoing email requests:

A alert tcp any any -> any 21 msg "Email sent"
B alert tcp any any -> any 25 msg "Email sent"
C alert tcp any 21 -> any any msg "Email sent"
D alert tcp any 25 -> any any msg "Email sent"
E alert tcp any 25 -> any 21 msg "Email sent"

2.14.3 Which Snort command will filter for incoming email from the server:

A alert tcp any any -> any 21 msg "Email received"
B alert tcp any any -> any 25 msg "Email received"
C alert tcp any 21 -> any any msg "Email received"
D alert tcp any 25 -> any any msg "Email received" "
E alert tcp any 25 -> any 21 msg "Email received" "

2.14.4 Which Snort command will filter for outgoing FTP requests:

A alert tcp any any -> any 21 msg "FTP out"
B alert tcp any any -> any 25 msg "FTP out"
C alert tcp any 21 -> any any msg "FTP out"
D alert tcp any 25 -> any any msg "FTP out"
E alert tcp any 25 -> any 21 msg "FTP out"

2.14.5 Which Snort command will filter for incoming FTP response from an FTP server:

A alert tcp any any -> any 21 msg "FTP response"
B alert tcp any any -> any 25 msg "FTP response"
C alert tcp any 21 -> any any msg "FTP response"
D alert tcp any 25 -> any any msg "FTP response"
E alert tcp any 25 -> any 21 msg "FTP response"

2.14.6 Which of the following is unlikely to be a port that a client uses to connect to an FTP server:

A 21
B 3100
C 3110
D 3111
E 4444

2.14.7 Which Snort command line option is used to define that packets are logged:

A -v B -c
C -n D -l
E -k

2.14.8 Which Snort command line option is used to read a rules file:

A -v B -c
C -n D -l
E -k

2.14.9 Which Snort command line option is used to run in verbose mode:

A -v B -c
C -n D -l
E -k

2.14.10 Which Snort command line option is used to define the log directory:

A -v B -c
C -n D -l
E -k

2.14.11 In Snort how might the home network variable to set

A var $HOME_NET=192.168.0.12\24
B var $HOME_NET 192.168.0.12\24
C $HOME_NET 192.168.0.12\24
D $HOME_NET=192.168.0.12\24
E var $HOME_NET is 192.168.0.12\24

2.14.12 In Snort, which is the default alert file

A alert.txt
B snort.txt
C myalerts.ids
D alert.ids
E source.alert

2.14.13 What does the "SYN", "SYN,ACK", "ACK" sequence signify

A The identification of a buffer overflow
B The retransmission of data
C The end of a client-server connection
D The handshaking of data for a client-server connection
E The initial negotiation of a client-server connection

2.14.14 For the "SYN", "SYN,ACK", "ACK" sequence, who generates the initial "SYN"

A The client
B The server
C Either the client or the server

2.14.15 For the "SYN", "SYN,ACK", "ACK" sequence, who generates the "SYN,ACK"

A The client
B The server
C Either the client or the server

2.14.16 For the "SYN", "SYN, ACK", "ACK" sequence, who generates the "ACK"

A The client

B The server
C Either the client or the server

2.16 Software tutorial

Snort is a useful program for implementing IDS, but it is rather general-purpose, and it can easily be over-burdened with high amounts of network traffic. This tutorial shows how it is possible to create a network sniffing agent, which can be built to process simple rules.

2.15.1 The WinPcap library can be used to read the source and destination IP addresses and TCP ports. For this the **TCPPacket** class is used. Initially modify the program in:

http://buchananweb.co.uk/srcSecurity/unit01_2.zip

so that it now displays the source and destination IP and TCP ports [4]:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is TCPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;

      TCPPacket tcp = (TCPPacket)packet;
      string srcIp = tcp.SourceAddress;
      string dstIp = tcp.DestinationAddress;
      int srcPort = tcp.SourcePort;
      int dstPort = tcp.DestinationPort;

      Console.WriteLine("{0}:{1} -> {2}:{3}", srcIp, srcPort, dstIp, dstPort);
   }
}
```

A sample run, using a Web browser connected to google.com gives:

```
84.53.143.151:80 -> 192.168.1.101:3582
84.53.143.151:80 -> 192.168.1.101:3582
192.168.1.101:3582 -> 84.53.143.151:80
```

where it can be seen that the Web server TCP port is 80, and the local port is 3582. Run the program, and generate some network activity, and determine the output:

Demo: http:// buchananweb.co.uk/media/unit02_1.htm

2.15.2 Modify the program in 2.15.1, so that it only displays traffic which is *distend* for a Web server. Prove its operation.

2.15.3 Next modify the code so that it detects only ICMP packets (using the ICMPPacket class), and displays the source and the destination addresses, along with the TTL (time-to-live) value [4]:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is ICMPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;

      ICMPPacket icmp = (ICMPPacket)packet;
      string srcIp=icmp.DestinationAddress;
      string dstIp=icmp.SourceAddress;
      string ttl=icmp.TimeToLive.ToString();

      Console.WriteLine("{0}->{1} TTL:{2}", srcIp, dstIp, ttl);
   }
}
```

A sample run is shown next for a ping on node 192.168.1.102:

```
Press any <RETURN> to exit
192.168.1.101->192.168.1.102 TTL:128
192.168.1.102->192.168.1.101 TTL:128
192.168.1.101->192.168.1.102 TTL:128
```

Run the program, and ping a node on the network. What is the output, and why does it show a number responses for every ping:

2.15.4 Modify the program in 2.15.3, so that it displays the Ethernet details of the data frame, such as [4]:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if( packet is EthernetPacket )
   {
      EthernetPacket etherFrame = (EthernetPacket)packet;
      Console.WriteLine("At: {0}:{1}: MAC:{2} -> MAC:{3}",
         etherFrame.PcapHeader.Date.ToString(),
         etherFrame.PcapHeader.Date.Millisecond,
         etherFrame.SourceHwAddress,
         etherFrame.DestinationHwAddress);
   }
}
```

2.15.5 It is possible to read the contents of the data packet by converting it to a byte array (using the Data property), and then convert it to a string, such as:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is TCPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;
      TCPPacket tcp = (TCPPacket)packet;
      byte [] b = tcp.Data;
      System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();
      string s = format.GetString(b);
```

```
            s=s.ToLower();
            if (s.IndexOf("intel")>0) Console.WriteLine("Intel found...");
         }
      }
```

The above code detects the presence of the word Intel in the data packet. Run the program, and then load a site with the word Intel in it, and prove that it works, such as for:


```
Intel found...
Intel found...
```

2.15.6 It is then possible to filter for source and destination ports, and with source and destination addresses. For example, the following detects the word Intel on the destination port of 80:

```
      private static void device_PcapOnPacketArrival(object sender, Packet packet)
      {
         if(packet is TCPPacket)
         {
            DateTime time = packet.PcapHeader.Date;
            int len = packet.PcapHeader.PacketLength;
            TCPPacket tcp = (TCPPacket)packet;
      3     int destPort = tcp.SourcePort;
            byte [] b = tcp.Data;
            System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();
            string s = format.GetString(b);
            s=s.ToLower();
            if (destPort==80 && (s.IndexOf("intel")>0))
               Console.WriteLine("Intel found in outgoing on port 80...");
         }
      }
```

2.15.7 A key indication of network traffic is in the TCP flags. The following determines when the SYN flag is detected, and also the SYN, ACK flags:

```
         if(packet is TCPPacket)
         {
            DateTime time = packet.PcapHeader.Date;
            int len = packet.PcapHeader.PacketLength;
            TCPPacket tcp = (TCPPacket)packet;
            int destPort = tcp.SourcePort;
            if (tcp.Syn) Console.WriteLine("SYN request");
            if (tcp.Syn && tcp.Ack) Console.WriteLine("SYN and ACK");

         }
```

Prove the operation of the code, and modify it so that it detects a SYN request to a Web server (port: 80), and displays the destination IP address of the Web server.

2.15.8 Write a program which displays each of the TCP flags, such as:

```
Packet flags: S----P--
Packet flags: SA------
Packet flags: -A------
Packet flags: -A-----F
```

Hint:

```
if (tcp.Syn) Console.Write("S") else Console.Write("-");
...
if (tcp.Fin) Console.WriteLine("F") else Console.WriteLine("-");
```

2.17 Snort tutorial

2.16.1 Determine the network interfaces of your machine with the snort –W option:

```
C:\Snort\bin> snort -W

Interface       Device          Description
-----------------------------------------
1 \Device\NPF_GenericNdisWanAdapter (Generic NdisWan adapter)
2 \Device\NPF_{3C369413-6967-4192-8CBC-203B57D95189} (Microsoft MAC Bridge)
3 \Device\NPF_{BD00EDD2-3753-4219-A043-F90108B30EEF} (NET IP/1394 Miniport)
4 \Device\NPF_{C215B0A7-CE88-424C-8669-D79264D5CF3E} (Intel(R) PRO/Wireless 2200)
```

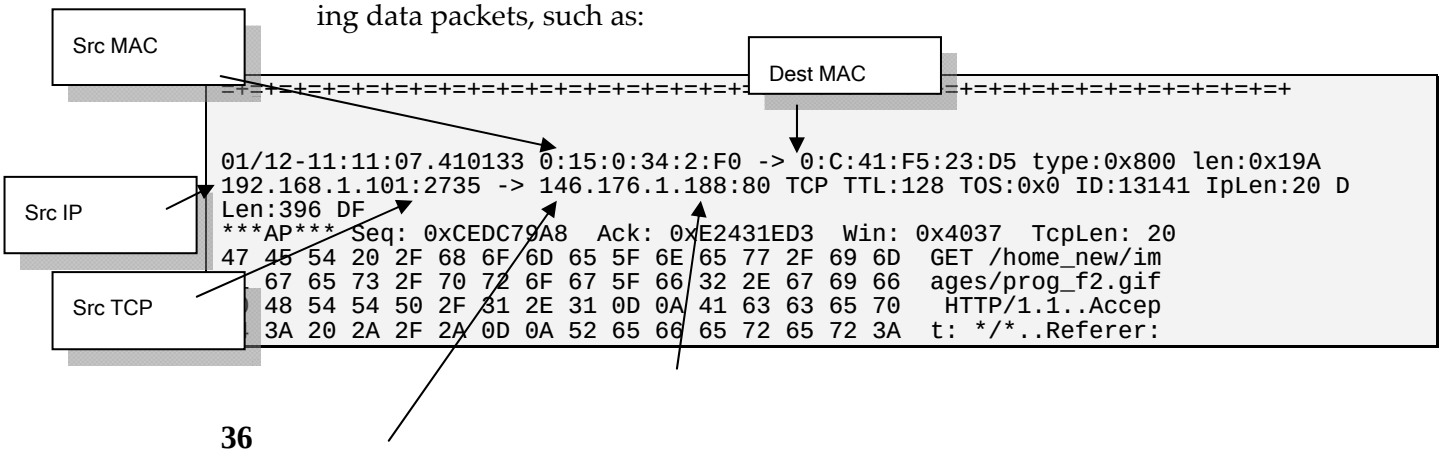
(a) Now run Snort with the interface that you want to use (in this case it is interface 4 – which is the Wireless interface). Note, if you are running Snort, you normally need to run it in promiscuous mode (which is default). On some network interfaces, such as the Intel 220BG the interface has a bug and the promiscuous mode is already set, so add a –p onto the command line:

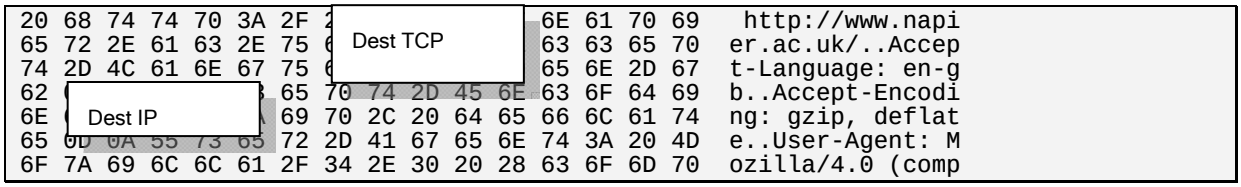
```
C:\Snort\bin> snort -dev -i 4 -K ascii
Running in packet dump mode

Initializing Network Interface \Device\
}

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

(b) Generate some Web traffic, and view the output, and verify that it is capturing data packets, such as:






```
20 68 74 74 70 3A 2F 2  6E 61 70 69   http://www.napi
65 72 2E 61 63 2E 75 6  63 63 65 70  er.ac.uk/..Accep
74 2D 4C 61 6E 67 75 6  65 6E 2D 67  t-Language: en-g
62 0  3 65 70 74 2D 45 6E 63 6F 64 69  b..Accept-Encodi
6E 6  A 69 70 2C 20 64 65 66 6C 61 74  ng: gzip, deflat
65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D  e..User-Agent: M
6F 7A 69 6C 6C 61 2F 34 2E 30 20 28 63 6F 6D 70  ozilla/4.0 (comp
```

(c) Select one of the TCP data packets, and determine the following:

The source IP address:

The source TCP port:

The destination IP address:

The destination TCP port:

The source MAC address:

The destination MAC address:

The TCP flags:

(d) Next run Snort so that it captures data packets and saves them to a subfolder. For this use the –l option to define the log folder, and –K to save as an ASCII dump (use **mkdir *YOURNAME***, to create a subdirectory, replacing *YOURNAME* with your name):

```
C:\Snort\bin> snort -dev -i 4 -p -l YOURNAME -K ascii
```

Access a few Web sites, and then stop the program, and examine the contents of your newly created folder:

What are the contents of the folder:

Go into one of the folders and view the contents of the IDS file. What does it contain:

(e) Next create a rules file which will detect the word "napier" in a data packet, for example:

```
alert tcp any any -> any 80 (content:"napier"; msg:"Napier detected"; sid:999)
```

(f) Save the file as napier.txt, and run the command, such as:

```
C:\Snort\bin>  snort -dev -i 4 -p -l log -K ascii -c napier.txt
```

(g) Access the Napier web site, and view some pages, and then go into your log folder and examine the alert.ids. Its format should be something like:

```
[**] [1:0:0] Napier detected [**]
[Priority: 0]
01/12-11:47:28.496017 0:15:0:34:2:F0 -> 0:C:41:F5:23:D5 type:0x800 len:0x171
192.168.1.101:3202 -> 146.176.1.188:80 TCP TTL:128 TOS:0x0 ID:15927 IpLen:20 Dgm
Len:355 DF
***AP*** Seq: 0x54962F22  Ack: 0x746ED796  Win: 0x44A8  TcpLen: 20

[**] [1:0:0] Napier detected [**]
[Priority: 0]
01/12-11:47:28.679437 0:15:0:34:2:F0 -> 0:C:41:F5:23:D5 type:0x800 len:0x175
192.168.1.101:3203 -> 146.176.1.188:80 TCP TTL:128 TOS:0x0 ID:15937 IpLen:20 Dgm
Len:359 DF
***AP*** Seq: 0xB7930606  Ack: 0x123ED8F3  Win: 0x44A8  TcpLen: 20
```

What is the contents of the alert.ids file:

Did it detect the word:

(h) Next download the client and server programs from (Figure 2.29):

↳http://buchananweb.co.uk/dotNetClientServer.zip

(i) In groups of two, one person should run the server on their computer, and the other person runs the client, and connects to the server on port **1001**. Make sure that you can chat, before going onto the next part of the tutorial.

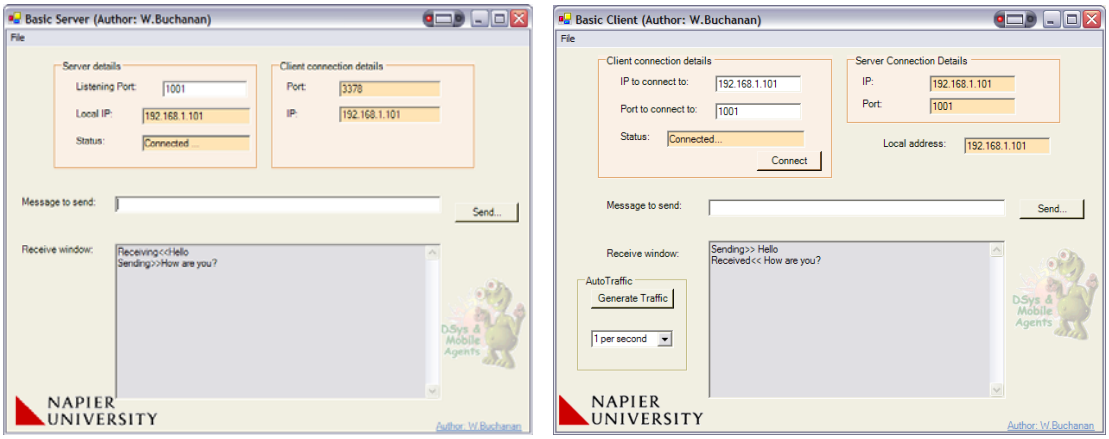


Figure 2.29 Client/server program

(j) Write a Snort rule which detects the word "napier" in the communications between the client and server.

What is the Snort rule for this:

2.16.2 Write rules which will detect the word **Intel** in the payload, for FTP, Telnet, MSN Messenger and HTTP, so that the alerts are:

Intel found in WWW traffic (port 80).

Intel found in Telnet traffic (port 23).

Intel found in FTP traffic (port 21).

Intel found in MSN Messenger traffic.

Verify your rules by running tests.

What are the rules:

(b) Run Snort, and monitor ARP the usage. From another host, ping a few of the hosts on the subnet, one at a time.

What do you notice from the ARP file during the ping process from the host?

... remember as you are connected to a switch, you will only see your own traffic, and any broadcast traffic, such as ARPs.

(c) A typical signature of a network attack is a port scan, where an intruder scans the open ports on a host. It is the objective of this lab to detect these types of attacks. Using Netstat, determine your connected ports, and using netstat –a, determine the all your listening port.

Connected ports:

Listing ports:

(d) A factor in security is to determine the TCP ports which are listening on hosts, as these are the way that an intruder can gain access to a host. Also it is possible to detect an intruder if they are scanning a network. Thus, down-load the NMAP portscanner. Note: DO NOT PORT SCAN ANY OTHER MACHINE THAN YOUR NEIGHBOUR'S COMPUTER. An example is at:

```
http://download.insecure.org/nmap/dist/nmap-3.95-win32.zip
```

A sample run is:

```
> nmap 192.168.1.1

Starting Nmap 3.95 ( http://www.insecure.org/nmap ) at 2006-01-12 13:26 GMT Standard Time
Interesting ports on 192.168.1.1:
(The 1668 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
80/tcp   open  http
8080/tcp open  http-proxy
MAC Address: 00:0C:41:F5:23:D5 (The Linksys Group)

Nmap finished: 1 IP address (1 host up) scanned in 2.969 seconds
```

Which ports are open:

Using the command netstat –a verify that these ports are open:

(e) Write a rule for Snort which allows a port scan to be detected, and verify that it works:

Snort rule:

Did it detect the port scan:

(f) Download the client and server program, and run the server on one machine and set its listening port to 1001. Rerun the port scanner from the neighbours machine.

↳http:// buchananweb.co.uk/dotNetClientServer.zip

Does the port scanner detect the new server port:

(g) Next with the server listing on port 1001. Now write a Snort rule which detects the incoming SYN flag for a connection from a client to the server.

What is the Snort rule:

2.18 Chapter Lecture

The additional material is at:

http://www.asecuritysite.com/security/information/chapter02

2.19 References

[1] http://www.snort.org

[2] http://www.faqs.org /rfcs/rfc821.html

[3] http://www.insecure.org/nmap/

[4] This code is based on the code wrapper for WinPcap developed by T.Gal [http://www.thecodeproject.com/csharp/sharppcap.asp].