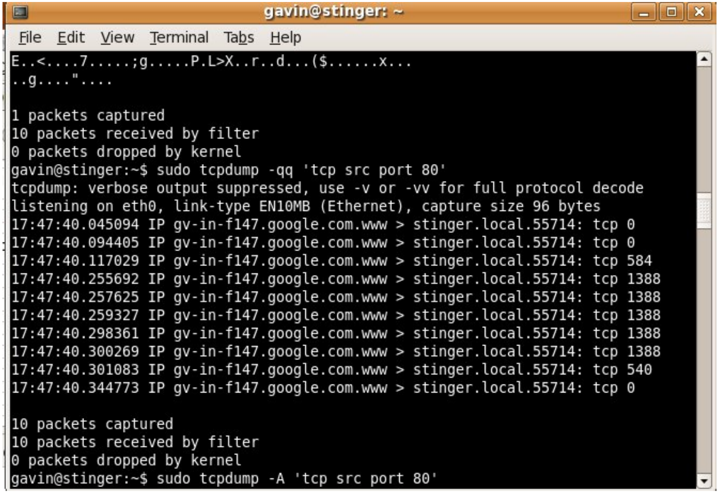
Packet sniffing

- On a hub LAN, packet sniffing is trivial as packets are broadcasted to all stations.
- The network card is put into *promiscuous* mode
- A packet sniffer is then used for example, wireshark or tcpdump.
- Anything transmitted in plaintext is trivial to observe
- Complete objects can be extracted from observed streams including images, executables, anything transmitted over the network.

Tcpdump output

Output of connection wget request to google.com



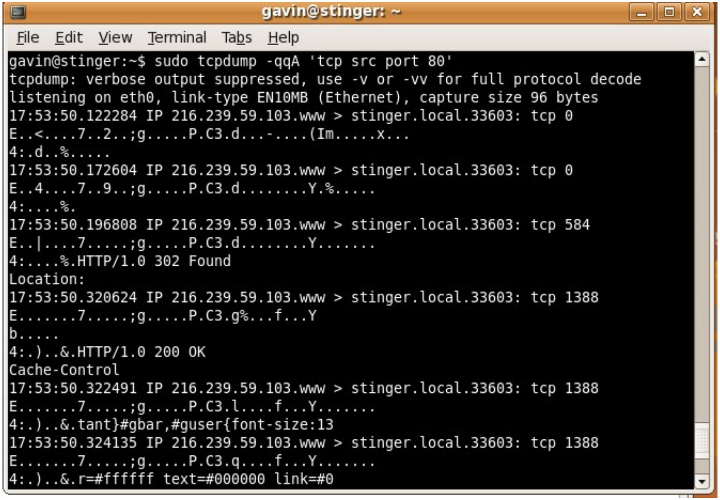
```
E..<....7.....;g.....P.L>X..r..d...($......x...
..g....".....

1 packets captured
10 packets received by filter
0 packets dropped by kernel
gavin@stinger:~$ sudo tcpdump -qq 'tcp src port 80'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:47:40.045094 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 0
17:47:40.094405 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 0
17:47:40.117029 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 584
17:47:40.255692 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 1388
17:47:40.257625 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 1388
17:47:40.259327 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 1388
17:47:40.298361 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 1388
17:47:40.300269 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 1388
17:47:40.301083 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 540
17:47:40.344773 IP gv-in-f147.google.com.www > stinger.local.55714: tcp 0

10 packets captured
10 packets received by filter
0 packets dropped by kernel
gavin@stinger:~$ sudo tcpdump -A 'tcp src port 80'
```

Tcpdump output

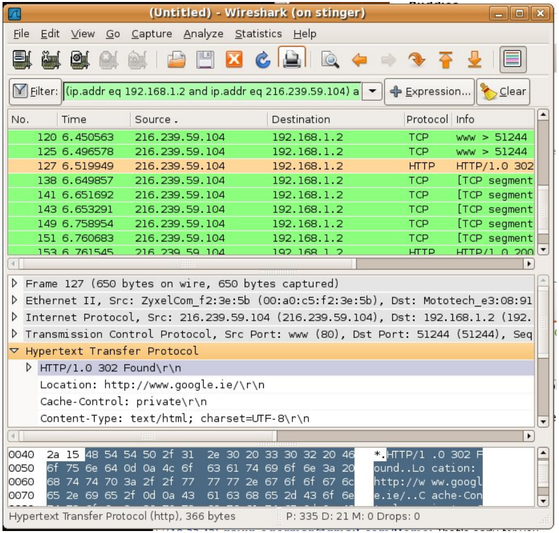
But we might want to see the actual payload. Use the man page to find out tcpdump options.



```
gavin@stinger:~$ sudo tcpdump -qqA 'tcp src port 80'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:53:50.122284 IP 216.239.59.103.www > stinger.local.33603: tcp 0
E..<....7..2..;g.....P.C3.d...-....(Im.....x...
4:.d..%.....
17:53:50.172604 IP 216.239.59.103.www > stinger.local.33603: tcp 0
E..4....7..9..;g.....P.C3.d.......Y.%.....
4:....%.
17:53:50.196808 IP 216.239.59.103.www > stinger.local.33603: tcp 584
E..|....7.....;g.....P.C3.d.......Y.......
4:....%.HTTP/1.0 302 Found
Location:
17:53:50.320624 IP 216.239.59.103.www > stinger.local.33603: tcp 1388
E.......7.....;g.....P.C3.g%...f...Y
b.....
4:.)..&.HTTP/1.0 200 OK
Cache-Control
17:53:50.322491 IP 216.239.59.103.www > stinger.local.33603: tcp 1388
E.......7.....;g.....P.C3.l....f...Y.......
4:.)..&.tant}#gbar,#guser{font-size:13
17:53:50.324135 IP 216.239.59.103.www > stinger.local.33603: tcp 1388
E.......7.....;g.....P.C3.q....f...Y.......
4:.)..&.r=#ffffff text=#000000 link=#0
```

Wireshark

Analysing a packet dump with tcpdump can be difficult. If possible, use wireshark, formerly ethereal.


(Untitled) - Wireshark (on stinger)
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear

Filter: (ip.addr eq 192.168.1.2 and ip.addr eq 216.239.59.104) a

```
No.  Time        Source .         Destination    Protocol Info
120  6.450563    216.239.59.104   192.168.1.2    TCP      www > 51244
125  6.496578    216.239.59.104   192.168.1.2    TCP      www > 51244
127  6.519949    216.239.59.104   192.168.1.2    HTTP     HTTP/1.0 302
138  6.649857    216.239.59.104   192.168.1.2    TCP      [TCP segment
141  6.651692    216.239.59.104   192.168.1.2    TCP      [TCP segment
143  6.653291    216.239.59.104   192.168.1.2    TCP      [TCP segment
149  6.758954    216.239.59.104   192.168.1.2    TCP      [TCP segment
151  6.760683    216.239.59.104   192.168.1.2    TCP      [TCP segment
153  6.761545    216.239.59.104   192.168.1.2    HTTP     HTTP/1.0 200
```

```
▷ Frame 127 (650 bytes on wire, 650 bytes captured)
▷ Ethernet II, Src: ZyxelCom_f2:3e:5b (00:a0:c5:f2:3e:5b), Dst: Mototech_e3:08:91
▷ Internet Protocol, Src: 216.239.59.104 (216.239.59.104), Dst: 192.168.1.2 (192.
▷ Transmission Control Protocol, Src Port: www (80), Dst Port: 51244 (51244), Seq
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.0 302 Found\r\n
    Location: http://www.google.ie/\r\n
    Cache-Control: private\r\n
    Content-Type: text/html; charset=UTF-8\r\n
```

```
0040  2a 15 48 54 54 50 2f 31  2e 30 20 33 30 32 20 46   *.HTTP/1 .0 302 F
0050  6f 75 6e 64 0d 0a 4c 6f  63 61 74 69 6f 6e 3a 20   ound..Lo cation:
0060  68 74 74 70 3a 2f 2f 77  77 77 2e 67 6f 6f 67 6c   http://w ww.googl
0070  65 2e 69 65 2f 0d 0a 43  61 63 68 65 2d 43 6f 6e   e.ie/..C ache-Con
```

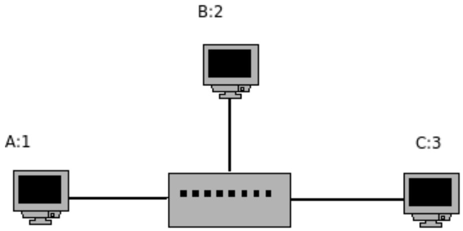
Hypertext Transfer Protocol (http), 366 bytes   P: 335 D: 21 M: 0 Drops: 0

Switched Networks

- Switched networks do not send all traffic to all ports, so sniffing is not possible. Or is it?
- By exploiting the Address Resolution Protocol, it is possible to sniff all, or just some traffic on a switched network. ARP Caches can be poisoned.
- Send the wrong data to other hosts on the network.

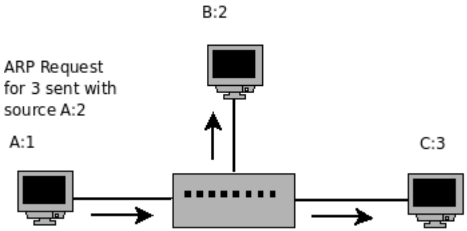
ARP Poisoning

We have a LAN with three machines connected via a switch. The machines have MAC addresses of A, B and C and IP addresses of 1, 2 and 3. A is the attacker. B and C are innocent parties. A wishes to see the communications between A and B.



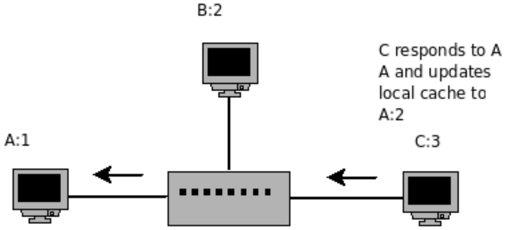
ARP Poisoning

A will create an Ethernet frame containing an ARP REQUEST for IP address 3. The ARP REQUEST will contain, as the source address A with a false associated IP of 2. This ARP REQUEST is broadcasted to the network.



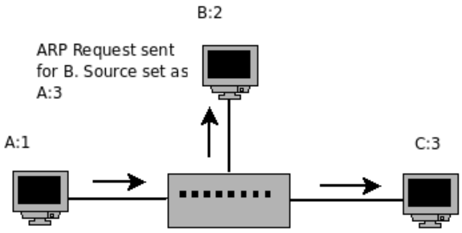
ARP Poisoning

C sees the ARP REQUEST and responds with its Ethernet address. At the same time, C updates its ARP cache with the source IP and MAC in the spoofed packet. C's cache now contains 2:A.



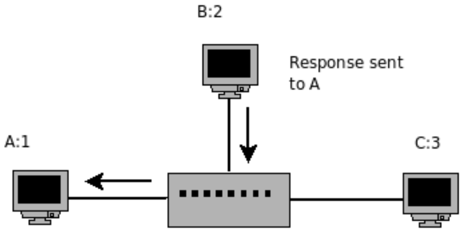
ARP Poisoning

A performs the same trick, this time sending the ARP REQUEST to B and using A:3 as the source MAC and IP.



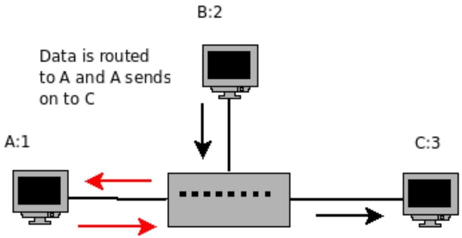
ARP Poisoning

B see the ARP REQUEST, replies and updates its cache with the false information. A has a cache with the correct mappings. B has a cache where IP address 3 points to MAC address A and C has a cache with IP address 2 pointing to MAC address A.



ARP Poisoning

B now wishes to connect to IP address 3. B sees 3:A in its cache, so sends the Ethernet frame encapsulating the IP packet to A. A can then choose to send the packet onto C having observed the contents, modify the contents, delay or delete. Any response from C is also sent to A.



Other ARP Related attacks

- Denial of Service - Send a false, non-existant MAC address for the local IP gateway to a target machine on the network. When this machine attempts to connect to IP addresses not on the LAN, the frames will be sent to nowhere. The machine will have lost external access.
- Overload a switch - The brute force approach to sniffing traffic on a switch. Send an excess of ARP updates to the switch. The switch memory will fill up and *may* drop down into basic hub mode, transmitting to all nodes

This is not an out of date attack

ARP poisoning, although quite old, is still a serious issue

- Wireless networks are a major victim
- Viruses and trojans can utlise ARP poisoning/spoofing on LANs
  - Assume you have a secure workstation on a LAN
  - Another machine on the LAN is not so secure and ends up with a virus on it
  - The compromised machine performs ARP poisoning implementing a Man in the Middle attack.
  - All connections from the secure machine are now routed via the in-secure. HTTP, SMTP, POP3 etc.
  - Attacker inserts malicious code into HTTP downloads and so installs a virus on the secure machine.

Prevention/Monitoring

How to prevent ARP Spoofing/Poisoning?

- On a small network, you could setup static IP:ARP mappings and disable updates. Not too feasible for a large number of machines though.
- If you have a managed switch, you may be able to restrict each port to a single MAC address
- You can monitor for suspicious ARP activity using ARPWatch. ARPWatch builds a database of MAC:IP mappings and notifies when there are changes.
- Authorization solely based on MAC address is a BAD idea. MAC addresses can be changed easily.

IP Spoofing

IP packets can be created manually and contain whatever their designer wishes.

- IP source address can be set to anything.
- Programmed using RAW Sockets, root only under unix.
- IP spoofing attacks will work with ICMP and UDP. TCP has sequence numbers which hinder the attack, but as we will see are not infallible...
- IP spoofing attacks are particularly dangerous when hosts use an IP address as authentication.
  - Older unix utilities, rsh, rlogin allowed access based on the source IP address.
  - Replace by SSH, discussed later

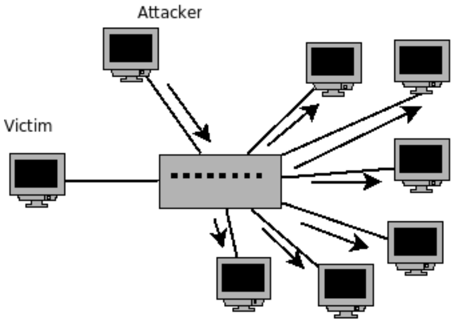
IP Spoofing

Basic ICMP attacks can be implemented using spoofing. A smurf attack is a Denial of Service attack where a false request will generate a storm of responses to a target address, overloading it. There are ICMP implementation of this attacks.

- A basic attack may be to send a stream of ping packets to a host in an attempt to waste bandwidth or processing time. The attacker can spoof the source IP address so as to not be identified. This is limited to the bandwidth the attacker has available to them.
- A more effective attack is to broadcast a ping to the entire network, with the source of the ping set to the victim. Any machine on the network configured to respond to ICMP broadcast pings will reply, potentially overwhelming the victim.

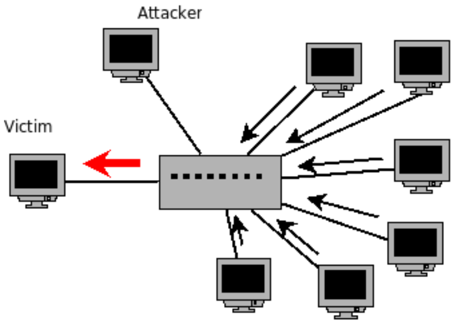
ICMP Spoofing: Ping attack

Attacker sends a single ping, ICMP ECHO request, packet to the broadcast address with the source as the victims address



ICMP Spoofing: Ping attack

The machines on the network respond with ICMP ECHO responses to the victim.

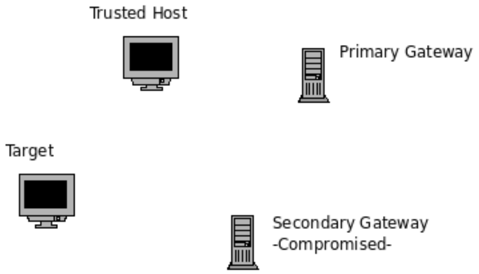


ICMP Spoofing: Redirect attack

The ICMP Redirect message is used to by gateways to advise of a better route. A Redirect message must be sent in relation to an existing connection however. That is, a client must have made a request for the Redirect to be sent.

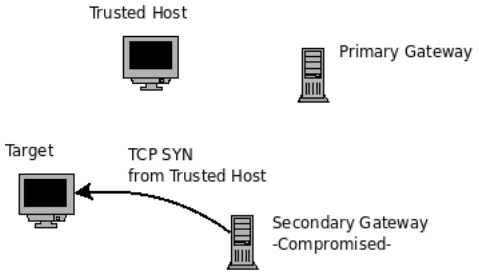
ICMP Spoofing: Redirect attack

Below we have two gateways, a primary and secondary. The secondary gateway has been compromised by an attacker. There is a target machine the attacker wishes to access and a trusted host. The target machine allows the trusted host to connect to it.



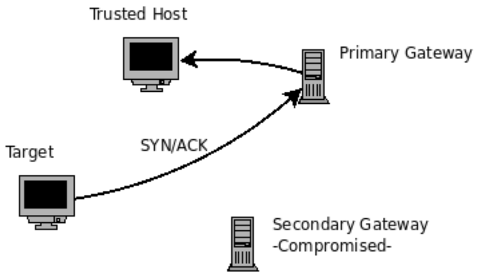
ICMP Spoofing: Redirect attack

The attacker sends a spoofed TCP SYN packet to the target machine. The spoofed source is the trusted host.



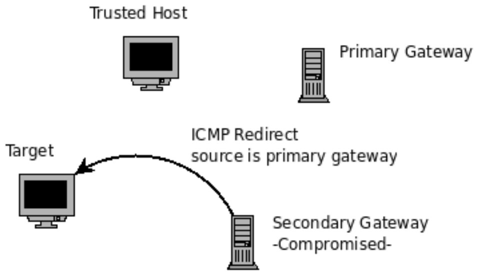
ICMP Spoofing: Redirect attack

Target machine responds with the SYN/ACK packet, routed through the primary gateway.



ICMP Spoofing: Redirect attack

The attacker now sends an ICMP Redirect packet to the target machine. The source is set as being the primary gateway and the Redirect message tells the target to use the secondary gateway in future. The target machine updates its routing table to reflect this.



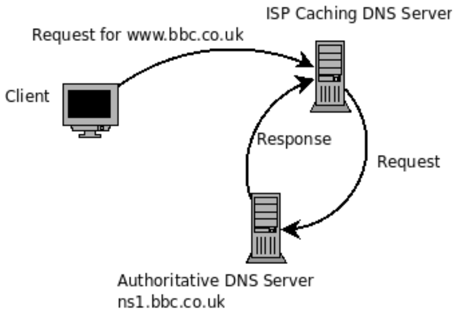
ICMP Spoofing

- ICMP attacks can also be used for additional Denial of Service attacks with the Destination Unreachable or Time Exceeded messages.
- However, practically ICMP attacks are not very common. RFC 1122 advises that most ICMP error messages by treated as advisory rather than mandatory.
- It is also the case that for LAN attacks, ARP poisoning can achieve the same results with less effort.

UDP Spoofing

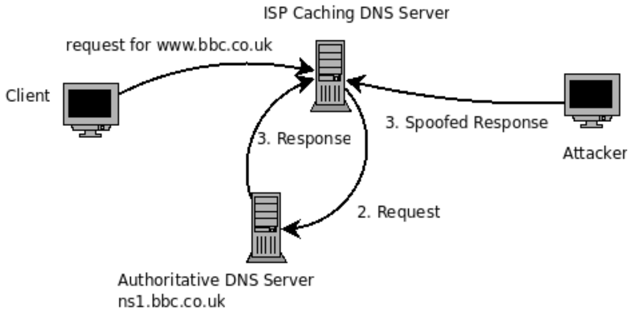
Applications using UDP may also be vulnerable to IP spoofing. DNS is one example.

- DNS requests and responses less than 512bytes use UDP.
- Request has a 2^{16} bit random ID number. The response from the server contains the same ID number as identification.
- Outgoing packet has a UDP source port.



UDP Spoofing: DNS

An attacker wants to replace the IP for www.bbc.co.uk with that of a machine they control. Solution is when the ISP DNS server issues a request to the authoritative server, spoof a response from that server with the attackers chosen IP.



UDP Spoofing: DNS

It's not quite that straightforward however when the attacker is not on the same LAN. The attacker cannot see the request, so does not know the ID, or the source UDP port or even the time the request is issued. The solution for the attacker is to:

- Issue the request for the domain him/herself.
- To discover the source port, issue some requests for domains the attacker controls, thus see what source port the server uses. The server may use the same port consistently.
- To discover the ID, the attacker uses a brute force approach of sending a large number of false responses with different IDs. With the Birthday Paradox, it actually evaluates to a .96 chance with 650 request/responses.

Kaminsky Attack

Dan Kaminsky found a more dangerous version of this attack. The attack is almost identical however, it allows an attacker to take over a complete domain, not just a single record.

1. The attacker makes a request for a random/unused subdomain in the particular domain he wishes to control.
2. The attacker then sends the forged response packets as in the previous attack, brute forcing the ID
3. However, instead of responding with the DNS record, the attacker sends a message delegating the response to another nameserver.
4. The attacker controls this nameserver and so now responds to requests for the compromised domain. MX, NS !

Kaminsky Attack

- With a sufficent amount of data, this attack can succeed in approximately 10 seconds.
- How to solve ?
-
    - Randomise the UDP source port as well
    - Potentially other solutions like DNSSec.

Preventing IP Spoofing

- One of the most important issues is to not accept packets claiming to be from inside your network, which originiate from outside the network
- ISPs should not route packets with addresses not on their network, but they often do. More of this in Distributed Denial of Service attacks.
- Unicast Reverse Path Forwarding - Don't route non-existant IPs and ensure that packets entering the router have a valid path to that router.

TCP Stream spoofing

Moving up to TCP, a new factor is introduced, that of the sequence numbers, negotiated during the TCP/IP handshake.
These sequence numbers make it difficult for an attacker to spoof a TCP/IP stream they cannot observe them.
However, TCP/IP implementations differ in how they generate the ISNs. Some implementations can be predictable!

Case Study!

An interesting case study detailed at

`http://www.gulker.com/ra/hack/tsattack.html`

Two o clock in the afternoon, Christmas Day, 1994, Kevin Mitnick breaks into Tsutomu Shimomura's network. He left mocking voicemails for Shimomura's. Shimomura is displeased and helps the FBI to track down and ultimately arrest Mitnick. Mitnick spent 5 years in jail. Mitnick used TCP/IP spoofing to access the network.
Shimomura's network consists of three machines

- server
- x-terminal
- target

The x-terminal computer trusts the server machine. That is, x-terminal is configured to allow connections from server.

Case Study

Mitnick begins with a reconnaisance of the network. His machine is toad.com. He is listing through information offered by the target, server and x-terminal machines.

```
14:09:32 toad.com# finger -l @target
14:10:21 toad.com# finger -l @server
14:10:50 toad.com# finger -l root@server
14:11:07 toad.com# finger -l @x-terminal
14:11:38 toad.com# showmount -e x-terminal
14:11:49 toad.com# rpcinfo -p x-terminal
14:12:05 toad.com# finger -l root@x-terminal
```

Case Study

Several minutes later, the server machine is bombarded with SYN packets from a non-existant IP address. The connection queue is filled and the server machine is unable to respond to any requests.

```
14:18:22.516699 130.92.6.97.600 > server.login: S 1382726960:1382726960(0) win 4096
14:18:22.566069 130.92.6.97.601 > server.login: S 1382726961:1382726961(0) win 4096
14:18:22.744477 130.92.6.97.602 > server.login: S 1382726962:1382726962(0) win 4096
14:18:22.830111 130.92.6.97.603 > server.login: S 1382726963:1382726963(0) win 4096
14:18:22.886128 130.92.6.97.604 > server.login: S 1382726964:1382726964(0) win 4096
14:18:22.943514 130.92.6.97.605 > server.login: S 1382726965:1382726965(0) win 4096
14:18:23.002715 130.92.6.97.606 > server.login: S 1382726966:1382726966(0) win 4096
14:18:23.103275 130.92.6.97.607 > server.login: S 1382726967:1382726967(0) win 4096
14:18:23.162781 130.92.6.97.608 > server.login: S 1382726968:1382726968(0) win 4096
```

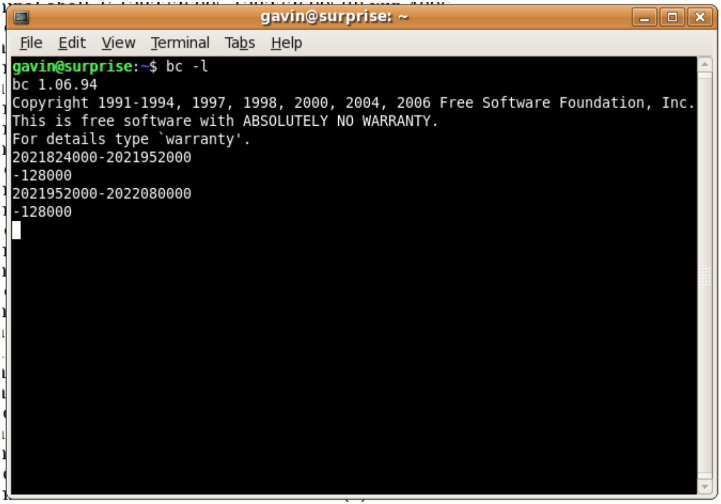
Case Study

20 connections are made from apollo.it.luc.edu to x-terminal

```
14:18:25.906002 apollo.1000 > x-terminal.shell: S 1382726990:1382726990(0) win 4096
14:18:26.094731 x-terminal.shell > apollo.1000: S 2021824000:2021824000(0) ack 1382726991
14:18:26.172394 apollo.1000 > x-terminal.shell: R 1382726991:1382726991(0) win 0
14:18:26.507560 apollo.999 > x-terminal.shell: S 1382726991:1382726991(0) win 4096
14:18:26.694691 x-terminal.shell > apollo.999: S 2021952000:2021952000(0) ack 1382726992
14:18:26.775037 apollo.999 > x-terminal.shell: R 1382726992:1382726992(0) win 0
14:18:26.775395 apollo.999 > x-terminal.shell: R 1382726992:1382726992(0) win 0
14:18:27.014050 apollo.998 > x-terminal.shell: S 1382726992:1382726992(0) win 4096
14:18:27.174846 x-terminal.shell > apollo.998: S 2022080000:2022080000(0) ack 1382726993
14:18:27.251840 apollo.998 > x-terminal.shell: R 1382726993:1382726993(0) win 0
14:18:27.544069 apollo.997 > x-terminal.shell: S 1382726993:1382726993(0) win 4096
14:18:27.714932 x-terminal.shell > apollo.997: S 2022208000:2022208000(0) ack 1382726994
14:18:27.794456 apollo.997 > x-terminal.shell: R 1382726994:1382726994(0) win 0
14:18:28.054114 apollo.996 > x-terminal.shell: S 1382726994:1382726994(0) win 4096
14:18:28.224935 x-terminal.shell > apollo.996: S 2022336000:2022336000(0) ack 1382726995
14:18:28.305578 apollo.996 > x-terminal.shell: R 1382726995:1382726995(0) win 0
14:18:28.564333 apollo.995 > x-terminal.shell: S 1382726995:1382726995(0) win 4096
14:18:28.734953 x-terminal.shell > apollo.995: S 2022464000:2022464000(0) ack 1382726996
14:18:28.811591 apollo.995 > x-terminal.shell: R 1382726996:1382726996(0) win 0
14:18:29.074990 apollo.994 > x-terminal.shell: S 1382726996:1382726996(0) win 4096
14:18:29.274572 x-terminal.shell > apollo.994: S 2022592000:2022592000(0) ack 1382726997
```

Case Study

If we look at the Initial Sequence Numbers being sent from x-terminal, we see something interesting



```
gavin@surprise:~$ bc -l
bc 1.06.94
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2021824000-2021952000
-128000
2021952000-2022080000
-128000
```

Case Study

Now that the ISN sequence is know, it is possible for Mitnick to actually establish a fake connection to the x-terminal machine from the server machine.

- A spoofed SYN packet is sent to x-terminal, from server
- x-terminal responds with a SYN, ACK packet to server
- But remember! Server was overloaded at the start, so is unable to respond with a RST packet.
- A spoofed ACK packet is then sent to x-terminal, with the guessed ISN.
- A connection has been opened.
- A spoofed data packet is then sent to x-terminal. The contents of it are

```
14:18:37 server# rsh x-terminal "echo + + >>/.rhosts"
```

Case Study

- Mitnick can now connect to the x-terminal machine from any other machine, as any user, there is no need for more stream spoofing.
- RST packets are sent to the server machine to clear the backlog and allow it to function normally
- The spoofed connection is closed.
- An existing connection from the x-terminal machine to the target machine is hijacked and Mitnick has achieved his goal.

That was 1994 !

That was a long time ago. Vendors have improved their TCP/IP stacks to use properly random ISNs. Or have they?

- CERT advisory CA-2001-09 demonstrates that Alcatel equipment, at the time, didn't use random ISNs, but increased the ISN by 64,000 every milisecond.
- In 2004, Watson discovered that RST packets were honoured, if the sequence number was anywhere within the recieve window. Window sizes are 32k or more, so less than 2^{17} guesses to generate a correct packet.

SSL/TLS Re-negotiation flaw

An interesting flaw was discovered in the TLS protocol recently, around November of 2009. The flaw allows a MITM attacker to prepend a conversation with a small amount of data.

- The innocent user attempts to connect to a target server, secured by TLS.
- The attacke has put himself or herself inbetween the client and the server, thus recieving the initial handshake values from the client.
- The attacker then makes its own connection to the TLS server. It sends a small amount of data and then requests a re-negotiation, sending on the original client parameters.
- The server and client establish a secure, confidential connection which the attacker cannot compromise.
- However. As part of the re-negotiation protocol, the server takes the initial piece of data from the attacker and prefixes it to the data sent by the client.

SSL/TLS Re-negotiation flaw

A practical, proof of concept implementation of this attack was used to compromise Twitter accounts.

- The attacker sends GET or PUT request with no CRLF.
- The clients header is then partially ignored, the initial Get header
- Further data, cookies for examples, are included.
- For Twitter, an update can be done through a simple HTTP PUT "twitter.com/statuses/update.xml"
- Attacker a status update msg to twitter with the attacker username/pwd
- When the client submits a status update, it includes the username/password. The full HTTP header is written to the attackers status.